Copyright
by
Sreejitha Vijayan
2013

The Report Committee for Sreejitha Vijayan
certifies that this is the approved version of the following report:

BusyBeeTaskManager:
Visual, Hosted, Collaborative Task Management

APPROVED BY

SUPERVISING COMMITTEE:

Adnan Aziz, Supervisor

Bruce McCann

BusyBeeTaskManager:
Visual, Hosted, Collaborative Task Management

by

Sreejitha Vijayan, B.E.

REPORT

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2013

To my husband Dr. Deepak Chandran for his relentless support and patience during these 2 years of combined school and work life.

Acknowledgments

I would like to thank my supervisor, Dr. Adnan Aziz whose guidance and vision made this report possible, and due to whom I accomplished much more in this report than I had initially set out for. Special thanks to Dr. Bruce McCann for graciously agreeing to be my reader. Thanks to all my professors at UT Austin, from whom I continually draw support and inspiration from.

BusyBeeTaskManager:
Visual, Hosted, Collaborative Task Management

Sreejitha Vijayan, M.S.E.
The University of Texas at Austin, 2013

Supervisor: Adnan Aziz

People today increasingly use software utilities which help create 'to-do' lists or reminders to stay efficient and productive. This has spurred a new industry into designing products for Business Management, Project Planning and Schedule Management on web and device platforms. However, these tools are either designed for novices or industrial level managers. A large number of individuals fall in between; their projects require intensive planning but not at the level of industrial projects. To address this category of users, I propose BusyBeeTaskManager, a web-based application that helps visually plan projects and efficiently communicate progress within the team. The user interface is designed for novice users but features such as document sharing, task dependency graph and automatic task updates allow users to carry out sophisticated team-based projects.

Table of Contents

List of Tables

List of Figures

Chapter 1

Goals

People are increasingly relying on software task manager utilities such as Mobile Calendars and 'to-do' lists to manage and disentangle their schedules. This basically stems from people's need to stay efficient and complete work on time. But the nature of most people's jobs today are so detail rich that simple to-do-lists cannot capture all the information about tasks or offer sufficient insight about the various properties of each task.

My main goal with this project was to build an application which not only manages schedules, but also provides a visual interpretation of the tasks at hand, while at the same time capturing dependencies and completion status. I wanted the user to be able to only fetch/modify information about projects, as was relevant to them, going into details only when deemed necessary. I wanted for team members to have a forum to put their comments in and be heard. Also, in terms of these forums, I wanted people to be able to share more than text in their reports and actually link files or pictures to their updates. Lastly, with this project I wanted to assign the application, rather than users themselves, the extremely time-consuming task of status information sharing, e.g., letting someone know that the dependent tasks have been completed.

1.1 Purpose statement: Make task management simple

The purpose of this project is to create a Web App, hosted on the cloud, that will allow a variety of people to manage tasks that have dependencies. The application will provide features that assist task scheduling and reporting between team members. The application will also facilitate user discussions, comments and sharing of files.

1.2 Problem: Hard to conceptualize or schedule complex projects

The support people need from technology is not always met by tools such as things-to-do lists. One primary reason for this inadequacy is due to the fact that many tasks depend on other tasks. While activities such as grocery shopping or paying bills can be handled by a simple list, projects involving teams often involve intricate dependencies between tasks. In team programming, team members often use a term called "blocking" to state that they cannot continue working as long as another dependency is not complete. In hindsight, many of these dependencies might appear to be obvious. The team should have been able to plan their tasks better in order to avoid blocking issues. It is not practical to predict all possible blocking conditions, but many of them are predictable. The team might have created plans where the dependencies were recorded. But the problem is that the dependencies were never recorded in a software tool that can assist the planning process. For example, if the dependencies were recorded in a task-manager that tracks them,

then that software tool will be able to inform the programmers about possible blocking situations that they should avoid.

Dependencies between tasks are not just applicable to software development. Other projects require the same type of support. For example, scientists performing lab experiments need to carry out the procedures in a specific order or sequence. When working in teams, this challenge is amplified because each team member must be aware of the tasks that the other team members are working on or have already completed. Similar to software engineering projects, researchers working in a team can also greatly benefit from tools that track task dependencies and task status. Hobbyists such as gardeners also have to think about non-trivial dependencies between tasks. Suppose a team is taking care of a community garden, then it is important for the team to know when and where new seeds were planted, which plants have received water and fertilizer and so on. The ordering of these tasks is essential, similar to software projects or scientific experiments.

There are several project management tools which allow team members and managers to list tasks, manage timelines, upload to code repositories and so on. However, unlike things-to-do lists, these software tools tend to be for large corporations or organizations. The cost and learning curve required for these corporate-level software tools make them less applicable to a variety of simpler use cases. For example, students working on programming projects for their homework assignments would not use such expensive and complex tools. Similarly, scientists who have little background in project management,

would not spend the time to learn such tools. Or, as mentioned earlier, people involved in projects done as a hobby such as gardening are unlikely to use corporate-level tools as well.

1.3 Vision: Assist users in collaborative, ordered tasks

Technology can aid people in areas where human memory is inadequate or provide services to perform tasks which they are forced to do repetitively as a function of their jobs. Giving or collecting status updates or assigning project tasks are examples of such functions. The fact that people still use direct communication techniques such as emails or phone calls to convey messages about day-to-day tasks implies that the current level of technology is insufficient to meet all the demands of task management. The purpose of technology is not to eliminate direct communication, but rather to complement it. Technology should make direct communication more fruitful by eliminating mundane questions such as "did you finish X?". In present day lifestyle, people often spend hours trying to arrange meetings with their team, reaching out to other people, sending back to back emails, only to understand or convey progress with tasks assigned.

To mitigate these problems I propose BusyBeeTaskManager, which eliminates the need to ask questions directly to the assignee of the task and get alerts to updates as they happen. BusyBeeTaskManager also allows for users to understand complete project task layout, dependencies and status with the help of visualization graphs. This website is also a collaborative platform to

discuss and post comments. As part of their updates, users can share files to demonstrate tangible results.

1.4 Overview: Chapters ahead

Chapter 2 will go over the project specifications and will have wireframes and description for the envisioned website look and feel. Chapter 3 will introduce various technologies used in the development of BusyBee. Chapter 4 will overview the system architecture and drill down into the actual implementation of envisioned goals. Chapter 5 will be a showcase on result metrics and link to demo video. Chapter 6 will summarize the report, compare BusyBee with related applications, and talk about future work. There will also be a section in this chapter on experiences and lessons learnt while building the application.

Chapter 2

Specifications and User Stories

This chapter goes over ideas and specifications, based on which the web pages and major features were designed and developed, with the initial mockups. Additionally, it will go over user stories, to give background and requirements that necessitated the development of these features.

2.1 Web Pages

My website would require to consist of a set of web pages, each serving an individual purpose to meet the vision and goals set for my application. Following are the web pages conceptualized at the time of design.

2.1.1 Task Manager Page

This page would have two sections which are bound together. The right hand side section is that of an editable grid, where the user can add, update or delete tasks and their corresponding high level details such as owner, dependencies, status and time-stamp. The left hand side section would be a node graph with edges to reflect the dependencies among tasks and color with timestamp, depicting task completion status. These two sections would

be needed to reflect the same data, hence if the grid data is updated, then corresponding changes would have to be made to the node graph. Figure 2.1 depicts a mockup of this page.

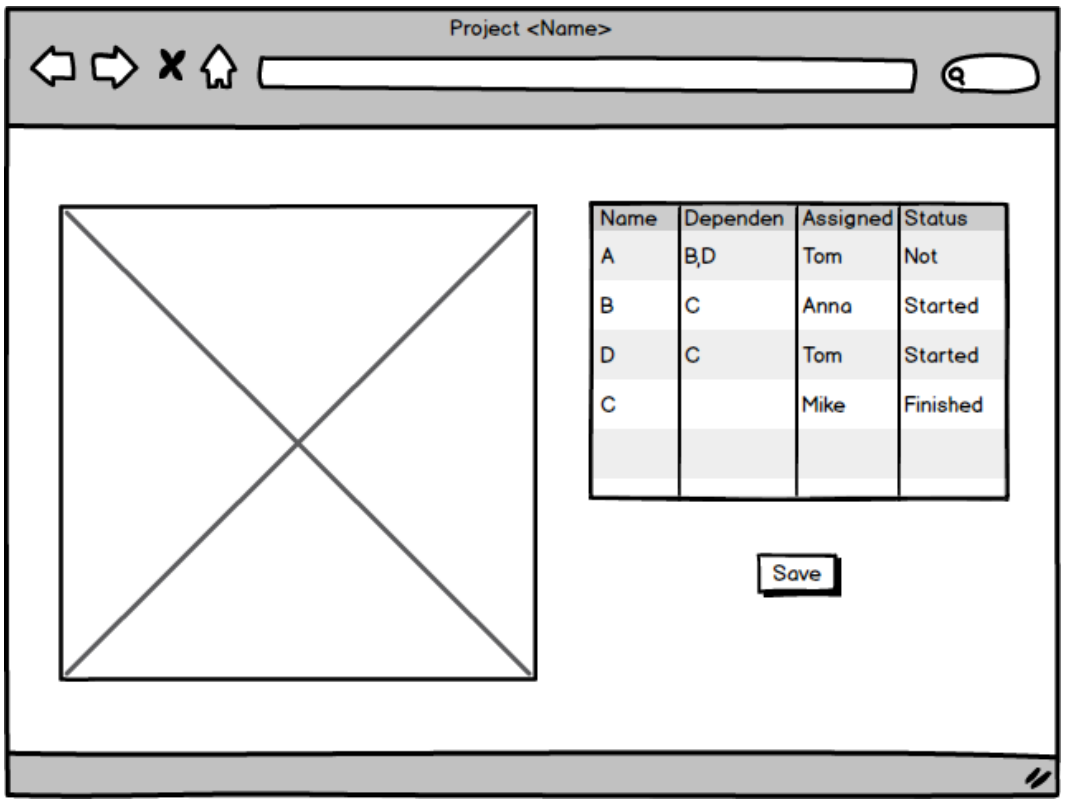

Figure 2.1: Task Manager Page with node graph canvas and editable grid with task listing for project

2.1.2 Task Details Page

This page would enable the user to give additional details about the individual task. To get to this page, the user would have to click on the corresponding node in the node graph of the Project Details page. These additional

details would be notes from the task owners or reviewers and document links supporting task status. This page would also provide a means to upload or share these documents. Additionally this page would have a discussion forum enabling project members to communicate about issues/challenges and ask questions.

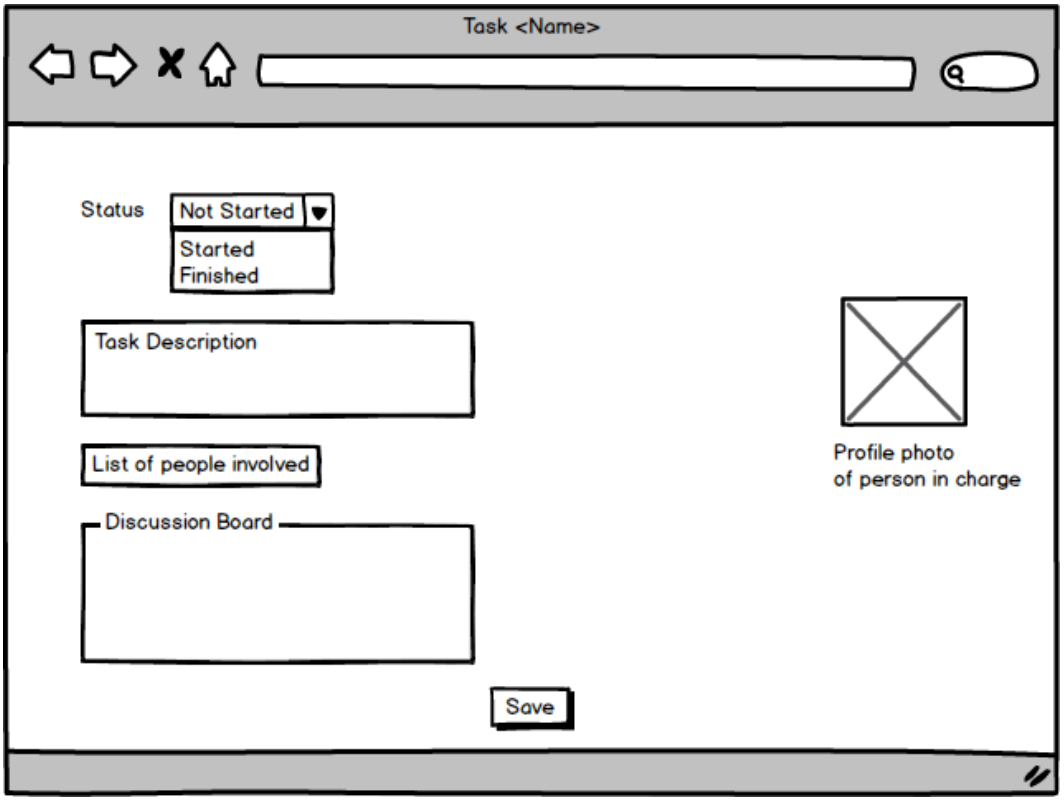

Figure 2.2: Task Details Page conveying status with detailed notes and a forum for discussion between team members

2.1.3 Start Project Page

This page would enable the user to start a new project and enlist users who are registered into BusyBee as team members. Figure 2.3, depicts a mockup of this page.

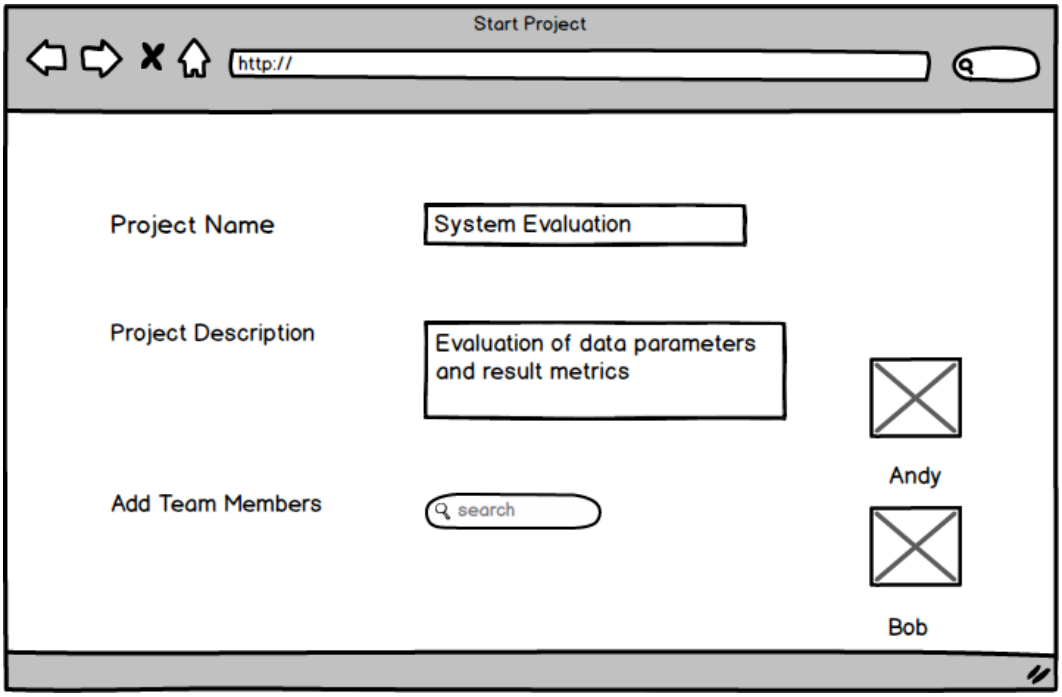

Figure 2.3: Start Project Page enabling users to plan a new project and recruit team members

2.1.4 View Projects Page

In this page, a registered user would be able to see high level details of all the projects that the user has initiated or has been invited to participate in. Figure 2.4, depicts a mockup of this page.

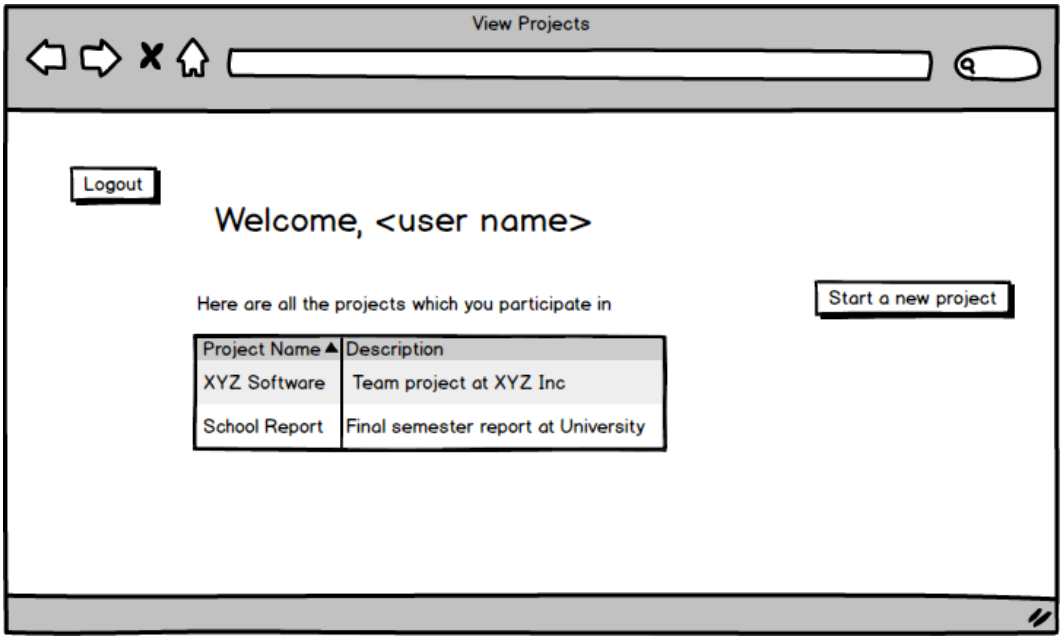

Figure 2.4: View Projects Page summarizing all of the user's active projects

2.1.5 Registration Page

This page would register a new user to the website. Contact information, photograph and a short self-summary would be sought from the user in this page. If the user is not registered to BusyBee, then this fact would be recognized in the first screen of the website post login and there would be a link to take the user to the Registration Page, where they would be successfully registered. Figure 2.5 depicts a mockup of this page.

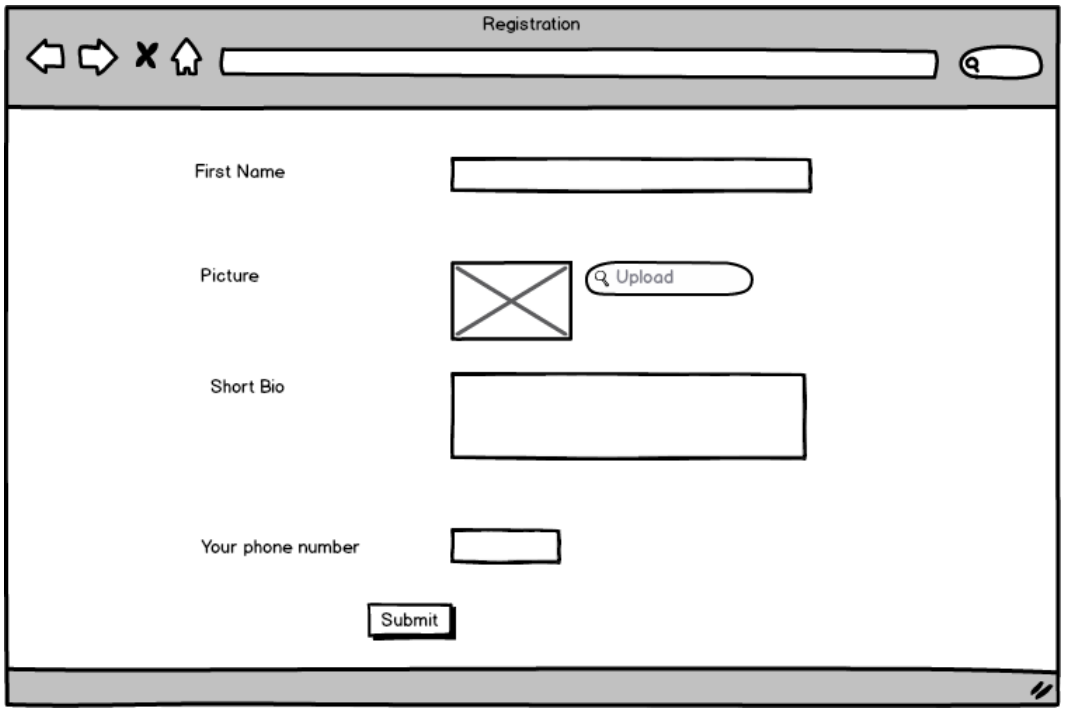

Figure 2.5: User Registration Page including fields required to register the user to the website

2.2 Features and Non Functional Requirements with User Stories

This section will talk about the major web features and Non Functional Requirements that were planned to be inculcated into BusyBeeTaskManager. With the help of user stories, it will also talk about why these features were considered important.

Visual

The core of this project was to enable users to plan tasks with visualization. The website needed to have node graphs which would diagrammatically represent the tasks of the project with sufficient detail. Each of these nodes would be a named task in the project. The connections between nodes would represent the task dependencies. Thus if there is an arrow from Task A to Task B, then it would mean that Task A needs to be completed before Task B is started. Some more details were need to be represented in the node graph. For example, each node would be labeled with the date and time of last edit. Thus if a user changed task status, or task notes, then the time of the latest edit would appear below the node in the node graph. Also each node would be a specific color to represent Task status.

Consider a user Jennifer, who is assigned a Task, '*Script for Event Handling*'. She is new to her project group and would have liked to know all the other tasks that either precede or succeed this task. In a traditional work assignment system, Jennifer would only focus on her task and would often wonder about the cause-effect relationship of her task with the other tasks. But having a node graph, would help her identify the 3 tasks that her task is dependent on. She would also identify 5 tasks, that depend on the completion of her task. Hence she would know, based on the node graph depicting completion statuses, when it is time for her to start working on her task. This enhanced understanding would enable her to better contribute to the working of her project and make it a bigger success.

Collaborative

The website was required to have forums which would encourage open communication among team members. These forums would also serve as an informal review or report for the corresponding task. Two instances were identified when communication among BusyBee users, would be necessitated. The first instance was that of Task Status update. The website would need a text editor, where text can be entered and also formatted.

Take a hypothetical BusyBee user Walter who wants to talk about progress made on a book which he is co-authoring. He needs to add a paragraph into a text editor as follows '*I have completed Chapter 19th of this report. Here is what I have focused on*'. At this point Walter would want to add some bullet points regarding what he has focused on in Chapter 19th. To do this, he would be able to use a button on a text editor, which would begin a bulleted list in his paragraph. Post this he would add his bullets and click 'Submit' button. Clicking on this button would create a new *post-it* note with the entered text into a *notes wall*. These are just a few of the features that the text editor would have to possess. Other notable features required would be addition of web links, converting the text into 'Bold' or 'Italic' format, ability to underline a sentence or word and ability to use several font colors and sizes. The above user story is an example for communication with regards to status reporting. The next user story, will outline communication with regards to Status Review.

Reconsider the user Walter, who has submitted his update for the task

'Writing Chapter 19th'. He needs approval from all of his team members before he finalizes the draft. Some of his team members cannot connect with him via phone, as they are traveling and only intermittently available. However Walter has to finalize his draft within the next few days. So the most efficient thing to do is get the team members of his project into a very brief discussion review. Thus Walter's traveling team members, would want to view his status update and give a 3-4 line feedback regarding improvements if any. In the website's discussion forum, Walter would be able to view and participate in a discussion among team members for each of his tasks, in each of his projects individually, to maintain relevance. This would equip Walter with clear action-items and goals post his update.

Another way of reviewing progress, would have to be through the *post-it* notes themselves. Consider a user Bob, who wants to make a suggestion to Walter regarding what he can additionally accomplish in Chapter 19th. He would be able to post a note on a *notes wall* saying,

```
Dear Walter,
Please work on some images to illustrate as well.
Thanks,
Bob
```

Thus Bob would be able to use *post-it* notes to review Walter's update which in turn accomplishes a much more informal, faster and effective way to

communicate among team members. He would also be able to participate in task discussions for each of the projects that he is a member of.

Data and File hosting

The website was required to be able to store data about the users, their work and their projects. This data would originate from web pages of the BusyBeeTaskManager website, in the process of registration, starting a project, task creation, task update or during user communication.

Thus, notes shared between Walter, Andy and other users, Walter and Andy's profile page information, a list of all of Bob's projects and all the messages in the discussion forum are examples of data needed to be stored into the website's database. The preference was to use a cloud data store, to eliminate the need for server costs for purchase and maintenance.

Shareable

The website was required to facilitate easy sharing of files created and maintained by the task owners with other team members. Thus there was a need to facilitate addition of web links of online files which a user would have worked on. But in case these files were not online, then the website would need a medium to upload and store these files by the task owners.

Editable

Since the users would be constantly working on their tasks which would mean that the tasks would continuously evolve with these edits, the website was required to enable easy edits to the user's task details. Thus the website was required to have forms and editable tables to facilitate the user to easily edit project and task details.

Consider a user Wilbert, who is formalizing a new project 'CRZ Software'. He would need to define a list of tasks needed to complete this project and would require an editable table where initial task data can be entered. He starts with adding three tasks to this table. Later he realizes that the third task is no longer a requirement, but in its place there is a new task, which he did not acknowledge before. In this case, Wilbert would need to have the ability to come back to this table and make the corresponding edits.

Notification enabled

With this project, I wanted to automate some of the counter-productive reporting tasks, that users perform in their projects, to allow them to focus on tasks that are more worth their time and attention. Hence it was required to set up an email and SMS service, to notify task owners when their dependent tasks have been completed.

Consider a user Kimberley, who is assigned a task 'Writing summary of contributions', which depends on the task 'Writing Chapter 19', which is assigned to Walter. Kimberley wants to begin as soon as she can, once the

dependencies are clear. However depending on Walter's timely update about task completion via email, is counter productive for both Walter and Kim. Therefore, as soon as Walter completes his task, the notification service would send a message to Kim on her registered phone number, saying that her task 'Writing summary of contributions', is clear for start. Along with the SMS, an email with the same message content would be sent to her gmail account, through this notification service.

Chapter 3

Technology Stack

This chapter will describe the software technology used in order to achieve the goals of this project. I have categorized the software components or frameworks used into logical sections based on its function.

3.1 Web App Framework

Writing a web application today has been simplified by frameworks which handle the common steps in web development for the framework user. Many of these frameworks are based on architectural patterns such as model-view-controller (MVC)[33] or the more traditional three-tier organization. Described below are the frameworks which formed the building blocks of my web application.

Web Hosting: Google App Engine

Google App Engine[8] (GAE) is a cloud based web hosting service provided by Google, to host web applications with a Google managed back-end. It supports several popular web programming languages such as Java, Python, PHP etc. Websites hosted can be monitored for data transactions, error logs,

usage statistics, status of cron jobs and many more similar application related non-functional requirement checks. GAE was the perfect choice to host BusyBeeTaskManager as the deployment completes in a few minutes and being cloud based, you can save on infrastructure costs. Over the course of development I started to weigh cost and effort versus advantage of switching over to Amazon Web Services[26] (AWS) or other services such as Heroku[31] or Rackspace Cloud[14]. My website did not need to perform resource intensive distributed computations such as Map Reduce or provide for authentication on top of the Google Sign In, or set up crawlers for search (which are some of the many services AWS wins over on GAE). I was looking at GAE purely for web-hosting initially and services that I wanted to later leverage such as Blob storage, Notification alerts and status monitoring were available for GAE as well. So, I decided to stick to GAE for web hosting.

My plan is to switch to the GAE paid version when the application data scales above 1 GB, but currently the free version suffices for my needs. Moreover the App Engine provided efficient data retrieval and update operations. Also, being able to check logs for errors in application deployment, was extremely useful for debugging purposes.

Data and Image Hosting: Google Blobstore and Datastore services

Apart from application hosting, GAE comes with a multitude of capabilities two such being the Google Blobstore[30] and DataStore[25]. These two services are available in the form of downloadable APIs that can be used in

the web application to query and perform transactions with a highly resilient data engine. Data including images are written into and retrieved from the data store as a collection of one or more entities. I used the Blobstore to store profile pictures of the users registered into the application. The Blobstore and Datastore can be used to store a variety of information. The former is used when the size of the object to be stored is much larger than the size allowed by the latter, typical in the case of images.

3.2 Visualization

Visual effects for web-based applications have gained tremendous amount of attention over the past few years, especially with the rise of Javascript apps and HTML5[36]. Google's "Chrome experiments"[29] are a great set of examples showing the potentials of Javascript.

GUI capabilities: jQuery plug-ins

jQuery[28] is a popular Javascript library, commonly used to traverse over, manipulate and animate web document elements. The library is free and open source under the MIT License. To quote the official jQuery Slogan "Write Less, Do More"[22]. jQuery allows for extensibility on top of any existing Javascript library in the form of plug-ins. The main motivation behind using jQuery was to free myself from handling widget behavior in different browsers which is a known problem for cross browser websites. So the option was to limit myself to fewer browsers or to use jQuery. Of course, I chose the latter

option. Today, it is difficult to find a large commercial website which has not leveraged jQuery for its user interface (UI). BusyBeeTaskManager uses jQuery for the function of date-picker and sticky notes. I will describe at length, these jQuery feature implementations in Chapter 3.

Raphael

Raphael[24] is one of the most popular vector graphics libraries for Web Apps in Javascript. Other popular Libraries such as Draw2D[32] are actually built using Raphael, demonstrating the flexbility of the Raphael library. Raphael uses Scalable Vector Graphics [37] (SVG). I began with another Javascript library which was like Draw2D, derived from RaphaelJS, called GraphDracula[38]. But I switched to its base library, Raphael, when I wanted more flexibility with the rendering of graphs. SVG is a standard format for storing 2D graphical objects such as lines, Bezier curves, circles, rounded rectangles and polygons made out of those basic shapes. SVG can also specify solid, transparent and gradient fill colors for polygons. I used Raphael to create project task dependencies graph. Several built-in capabilities exist within Raphael. For example, web developers can choose to add text below the node or directed line, position text with respect to the node, create animations and highlight nodes when they are hovered over.

SlickGrid

I needed a Data Grid framework for BusyBee, to enable users to view a list of their projects in progress, with a high-level description. But the Grid Framework mainly needed to support individual cell formatting, creation of custom controls, render thousands of rows with high speed along with standard features found in most *plug-in* grids such as data search and pagination. Based on several developer reviews and ratings[35][16][23] , I chose SlickGrid[34] for the View Projects page. SlickGrid is a Javascript library which had all the above mentioned features and many other useful features, such as 'GlobalEditorLock', which enables concurrent edits to the table and 'Undo/redo support' for cell editing. A key highlight of Slickgrid, which enables it to render data quickly is its property of virtual rendering which only fetches rows viewable in the data viewport causing rows (thousands of them) to be fetched and dropped with the scroll action of the user. This property is responsible for the high speed rendering in SlickGrid.

EditableGrid

In order for the user to be able to view as well as update task progress, which would dynamically change the visual task dependency graph, I needed a grid which the users could interact with. As with SlickGrid, I was looking for a customizable grid library which additionally enabled dynamic user edits. I considered Google Table Chart[20], but discovered that it would be challenging to add more functions to this library and tie it up with other features such

as the Visualization graph and also to make the cells editable. This is when I found out the EditableGrid[6] library. Some example of table cell to graph updates that I needed to perform are as follows:-

- If a new task A is added into EditableGrid, a corresponding new node A should be inserted into the Visualization graph
- If a new task B is dependent on task A, then a corresponding new node B should be inserted into the Visualization graph and an edge should be drawn between node A and node B, with the arrowhead pointing to node B.

Editablegrid is a Javascript library, aimed at providing *editable* html tables, as perhaps the name would suggest. The source of data could be anything from XML file to dynamic content or a database. The only downside with this framework is that, it neither recognizes nor formats the data, depending on type (URL, sql table column, caption), for the user. For my website, the data source is the Google Datastore. Today, more commonly, developers write code to fetch data into their grid in json format, which is what I did with Editablegrid. We will look at the interaction of Editablegrid and Raphael library in the next chapter.

TinyMCE Editor

TinyMCE[18] is a web-based editor for HTML text fields. It can be used for formatting text in the text fields for various effects such as bold,

italic, underline, ordered lists, unordered lists, image attachment etc. Such editors facilitate the end user to work on their documents online which means that the documents can be accessed from any device, anywhere. Several rich editors are available today to format documents such as Technical Specifications document or Resume. For each of the formatting effects mentioned above, there is a unique plugin that comes shipped with the TinyMCE library. I chose TinyMCE editor because its interface is much similar to the word processor interface that is used to edit documents, which in turn meant a seamless transition to the online editor for the end user.

3.3 Dropbox API

Within the last few years, saving files on the "cloud" has become common practice for many people. Ubuntu One, Dropbox, Google Drive and SugarSync are some examples of software tools that allow users to store files on their cloud-based storage facilities. Some of these tools, such as Dropbox, use other companies' cloud services, e.g., Amazon and forward the cost to users.

For programmers, Dropbox stands out as a unique cloud storage system because of its mature API. The Dropbox API[4] allows programmers to store and retrieve files from their software tool's private Dropbox account or even access files from their application user's Dropbox account, given an access permission. This feature allows programmers to create Web Apps that have the ability to store files on the cloud via the Dropbox API. I used Dropbox

API in this application and specifically the Dropbox Chooser and Dropbox file upload components for Java. However, it suffices to say that the API is available in other programming languages such as PHP, Ruby, iOS and Python. Recently the API Developers extended support to Javascript in lieu of the fact that entire applications are increasingly being written in Javascript today[27] . Dropbox chooser allows users to share files from their Dropbox account, whose web links are returned on select. With the Dropbox File Upload API, the user can use a file upload interface on the web page of an application registered in Dropbox, to upload files into their dropbox accounts.

3.4 Social component

Websites encouraging discussions and networking extremely have become popular over the years. This trend has caught on, so much so that many corporations built company wide social forums for employees to securely connect with each other. An example of this is Yammer[21], which is now offered as port of the Microsoft Office suite and Socialcast[7] by VMware. These social forums are becoming increasingly popular among businesses as it opens up an informal and unbounded platform for discussions without the time or space limitations of in-person meetings, events or social gatherings.

Disqus

Disqus[5] is a global commenting platform, which can be leveraged into websites. It has all the standard features of a commenting system, such as

ability to write posts or reply to them, share pictures, put ratings to posts, share post with others. Many large sites such as Tumblr use Disqus as their commenting platform. BusyBeeTaskManager uses Disqus for team members of a project to communicate their thoughts or decisions regarding the project with each other. Disqus also has an administration platform, where comments can be monitored and deleted when marked as inappropriate. Disqus uses the login credentials of the host website and does not need an additional Sign In process. Thus the host website itself (in my case BusyBeeTaskManager) does not have to store the comments in its servers, but instead Disqus manages and stores data for them in their own cloud storage system.

3.5 Notification Service

According to the definition in Wikipedia[13], notification services provide means to deliver automated messages to many persons at once. Such messages are event-based and also subscription-based. Two examples of 'event-based' would be '50% off on all Nike products' or 'Delay of Delta Airlines flight'. Thus a subscriber to Nike sale notifications will receive the notification of sale, only when the sale event condition turns true. Also, only a subscriber to Nike sale notifications, would in fact receive such a notification. These notifications can be sent via email, SMS etc. There are several popular notification services today available on the web and device platforms such as Twilio[19] for SMS and Google Cloud Messaging[9] (GCM) for Android push notifications.

SendHub SMS Notification Service

SendHub[15] is a free SMS Notification service that allow web application to send messages to registered mobile phone numbers in United States. BusyBeeTaskManager registration requires the users to enter their mobile phone number, which is used by SendHub to send SMS messages regarding various task status updates. BusyBeeTaskManager registers all the users to SendHub for them. I wanted an SMS Service which would be configurable for Java, or at best, directly have a version of its API written in Java. While both of them are extremely similar, Twilio lacked direct support for Java. That is why I chose SendHub over Twilio.

Technology	**Purpose**
Google App Engine	Web hosting, Data and Image hosting
Google Datastore	Part of the Google App Engine that deals with data hosting
Google Blobstore	Part of the Google App Engine that deals with image hosting
JQuery plug-ins	Provides several Javascript GUI features such as date picker, menu items and sticky notes
RaphaelJS	Javascript library used to create Bezier curves, Circles, and Tooltips for rendering the dependency graph
EditableGrid	Javascript library used to create table that is editable by the user. Callback functions are used to dynamically update the dependency diagram.
SlickGrid	Javascript library that creates the initial table showing all the projects that the user is a part of.
TinyMCE Editor	Web based HTML WYSIWYG Editor for text fields.
Dropbox API	Java API provided by Dropbox that allows uploading, downloading and sharing of files from the users' personal Dropbox accounts.
SendHub	web service that is used to send SMS messages to users regarding task updates.

Table 3.1: Summary of Technologies Used

Chapter 4

Implementation

The implementation of BusyBeeTaskManager was accomplished in a bottom-up heirarchy. Individual functions of the application were implemented in silo and then grouped into web pages, which were connected through navigation. Hence in this chapter, just like the process I followed for development, you will see that I have sectioned my implementation discussion based on web pages. I have described each web page as a sum of its functional components, followed by a visual representation of each these components individually or the web page in its entirety.

4.1 Task Manager Page

This web page provides for the addition, deletion, modification of tasks and their primary details. Users can visualize the task tree of their projects from the task graphs, which in turn is built from information entered into the editable task tables.

Editable grid integration with RaphaelJS

Editable Grid provides a callback called *valueChanged()* that allows a web application to perform any function when the user edits a value in the table. In the case of BusyBeeTaskManager, this callback capability is used to re-render the dependency graph. Re-rendering the entire graph during each change might seem inefficient, but the speed of rendering is fast enough that it is invisible for the human eye, even for relatively large (50+ tasks) dependency graphs.

Inside the callback function, BusyBeeTaskManager iterates through the Editable Grid values and creates a Javascript object (JSON compatible). The Editable Grid's columns are always in a specific order; for example, the first column is always the task name and the second is always the list of dependencies. Because the order of columns is consistent, it is possible to create a Javascript object from the columns. The Javascript object contains information on the tasks, their dependencies, contact person and so on. This Javascript object becomes the input to the RaphaelJS-based class called TreeLayout. The TreeLayout class takes the Javascript object and renders the nodes and connections.

Dependency Graph Auto-layout

As mentioned in the previous section, the auto-layout class (TreeLayout) takes a Javascript object as input. Below is an example of such a Javascript object. Nodes can be defined in any order - a parent node can

be defined after the child node. The algorithm will create all the nodes first before performing the layout and therefore, the order is not important.

```
'nodeC':
    {
      parents: ['nodeB','nodeA'],
      fill: '#f00',
      stroke: '#fff',
      radius: 6
    }
```

I created a function, *draw()* into the TreeLayout class which defines the way the node should appear (color-coded circle, with label below it) and also how a connection is to be established with other nodes. The *draw()* function also defines node behavior on events such as node selection and de-selection. When a node is hovered over, *nodeselected()* method is called, which increases the radius of the node, thereby making it bigger and also increases the width of the bezier curve connecting the selected node with the other nodes. *nodedeselected()* method changes the radius of the node and the width of the bezier curve back to its default values. There is one more key function in the TreeLayout Javascript file, called *calcdepth()*. This function is used to determine the placement of a newly added node, depending on the placement of its predecessors. A successor node will always appear to the bottom right of its

predecessor node. Also all nodes at the same horizontal level are seperated by a defined space value, so that they do not overlap.

However having a node graph growing bottom- rightwards, would mean that it would overlap with the Editable Graph at some point of time. Hence I allowed for the infinite expansion of the canvas but limited the visible area of the canvas to specific dimensions. To view the area of the canvas that is not visible, the right and bottom scroll bars can be used.

Editable grid integration with Google App Engine

The other important feature that Editable Grid provides is the ability to initialize the table from a normal HTML table. This feature allows BusyBeeTaskManager to load the dependency for a particular project from the datastore.

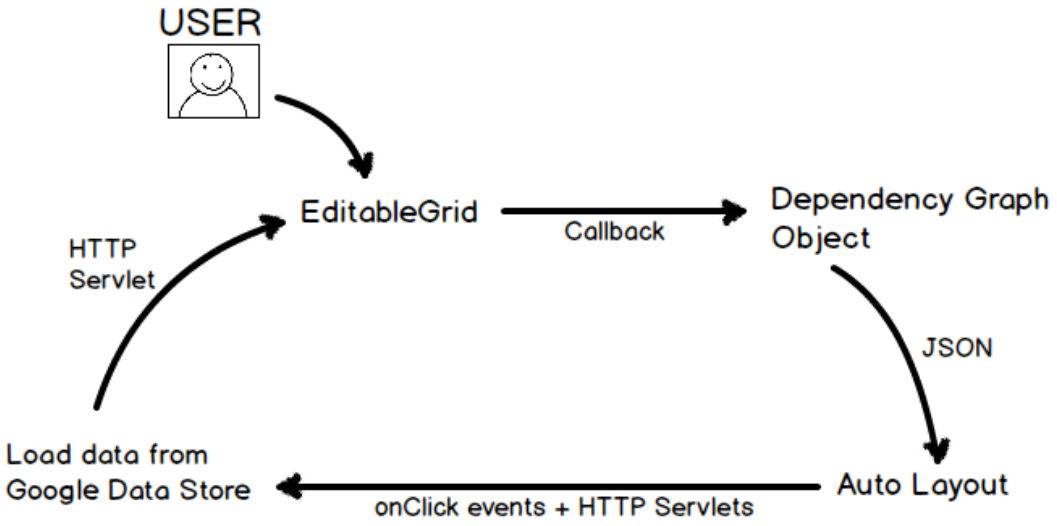

Figure 4.1: Task Manager Page - Event Flow

Figure 4.1 illustrates the flow of information involved in loading the project tasks from the server, populating the Editable Grid, updating the dependency graph and finally, saving any changes back the server. Figure 4.2 and Figure 4.3 display screenshots of the editable grid to create and/or edit task details and the task graph respectively. Notice that the arrows between the nodes in the task graph correspond to the dependencies between the tasks. The above mentioned components together form the Task Manager page in the website.

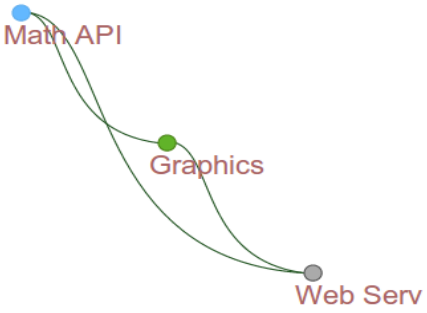

Figure 4.2: Task Manager Web page. This is the left-hand side which shows the relationships between tasks as a graph, with the most dependent tasks at the bottom-right.

Tasks list for Project Name

TASK NAME	DEPENDS ON	PRIMARY CONTACT	STATUS	TIME STAMP
Graphics	Math API	sreejitha.vijayan@gmail.com	Started	7/28/2013 @ 16:28 PM
Math API		andyandyking1986@gmail.com	Finished	7/28/2013 @ 16:28 PM
Web Serv	Graphics, Math API	andyandyking1986@gmail.com	Not Started	

Submit

View Projects

Figure 4.3: Task Manager Web page. This is the right-hand side of the web page which shows the editable table that is used to modify tasks.

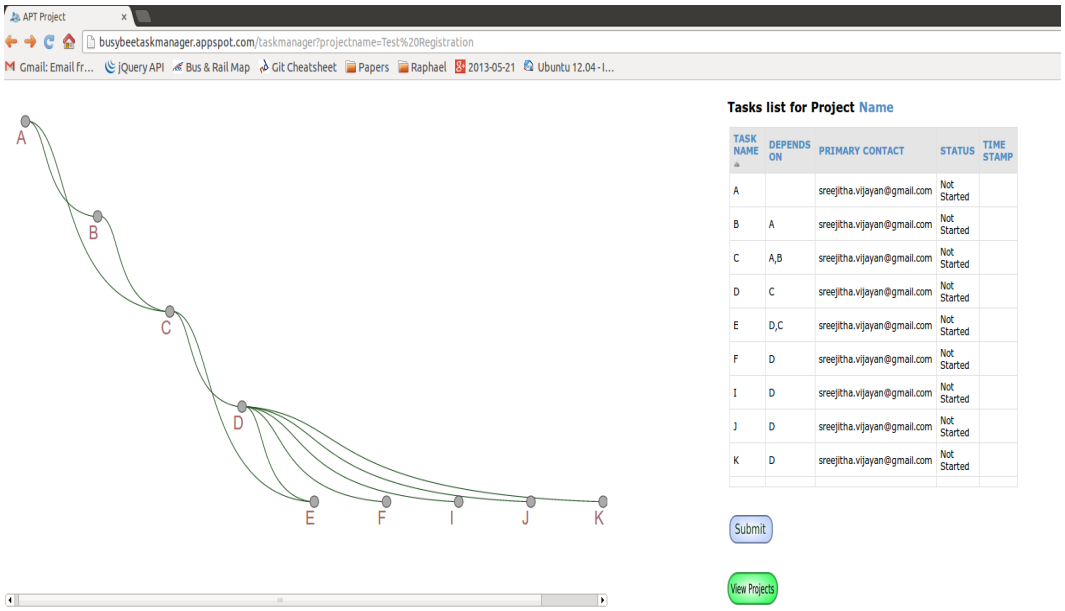

Figure 4.4: Task Manager Web page. The node graph canvas becomes scroll-enabled when nodes are added outside its viewable area.

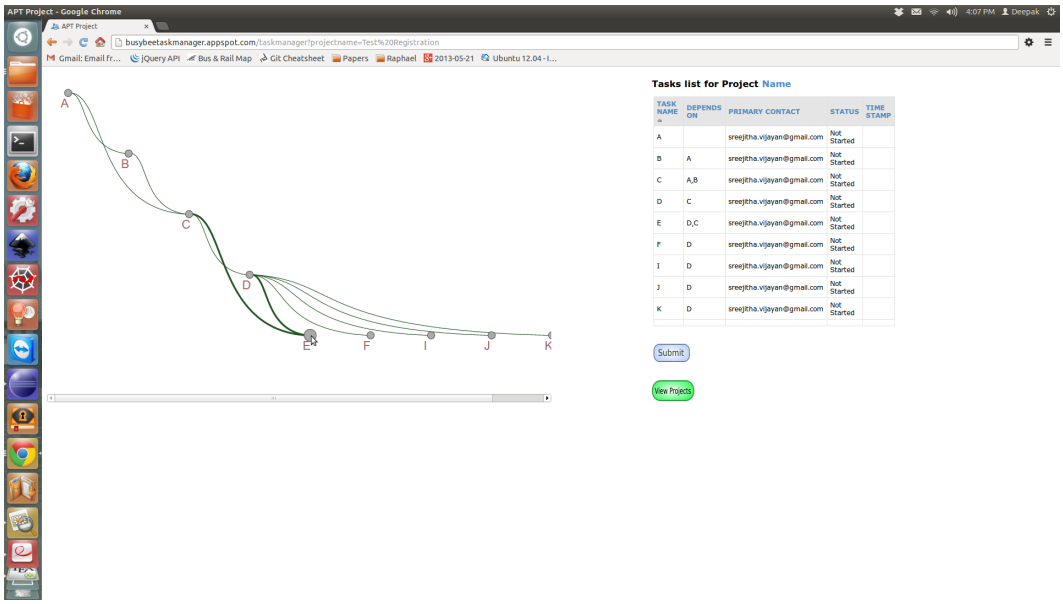

Figure 4.5: Task Manager Web page. The selected node (in this case: E) and its connections are highlighted when hovered over.

4.2 Task Details page

Task Details web page, as the name suggests, is about individual tasks in the project. As required by the specifications, this page facilitated the user to enter task status, support this status with documents, or even comment or engage in a dialog with other team members.

TinyMCE Implementation

The Editor interface has buttons for formatting features such as Bold, Italic and Underline. The click event of these buttons are tied to the TinyMCE Javascript event handler which then performs the formatting effects on the

text block. For each individual feature say 'Underline', there is a separate Javascript file, which handles functions specific to that feature.

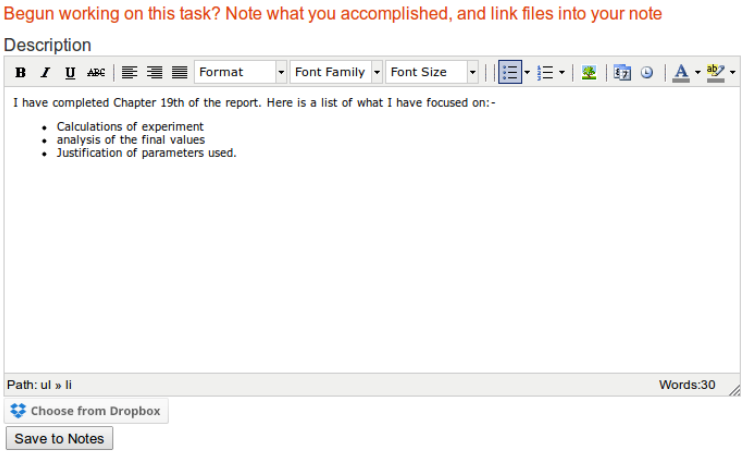

Figure 4.6: Tinymce Editor on Task Details Page

Let us go over one such function in the file corresponding to the function 'underline'. The *update()* function obtains the web document object, from which it obtains the form and the width parameter of the form. Using this, it generates an HTML snippet, with one element <hr> corresponding to horizontal rule. This HTML snippet is then appended to the editor interface.

Just a few of the buttons of the entire set that needed to appear on the editor were chosen and others such as 'Print' and 'Save' were removed. I retained 'Link', which can be used to add pictures into the text block. All the entries in the editor interface, get converted into a *post-it* note on the click of 'Add to Notes'. A snapshot of the editor interface can be seen in Figure 4.6.

Sticky Notes feature Implementation

In the Javascript file for this library, there is a function to add a new sticky note of default size (200,200) and no text. I added a new function called *createandrendernote()* to add text into this empty note. This text is taken from TinyMCE Editor interface. So, I tied the text returned from the click event of the editor to *createandrendernote()*. Thus every time the 'Save to Notes' button is clicked, a new sticky note is created with the entered text. The text is passed along with being rendered in html format. Also a user can add infinite sticky notes to the board, as I have made the board to have infinite scroll on both the horizontal and vertical axes.

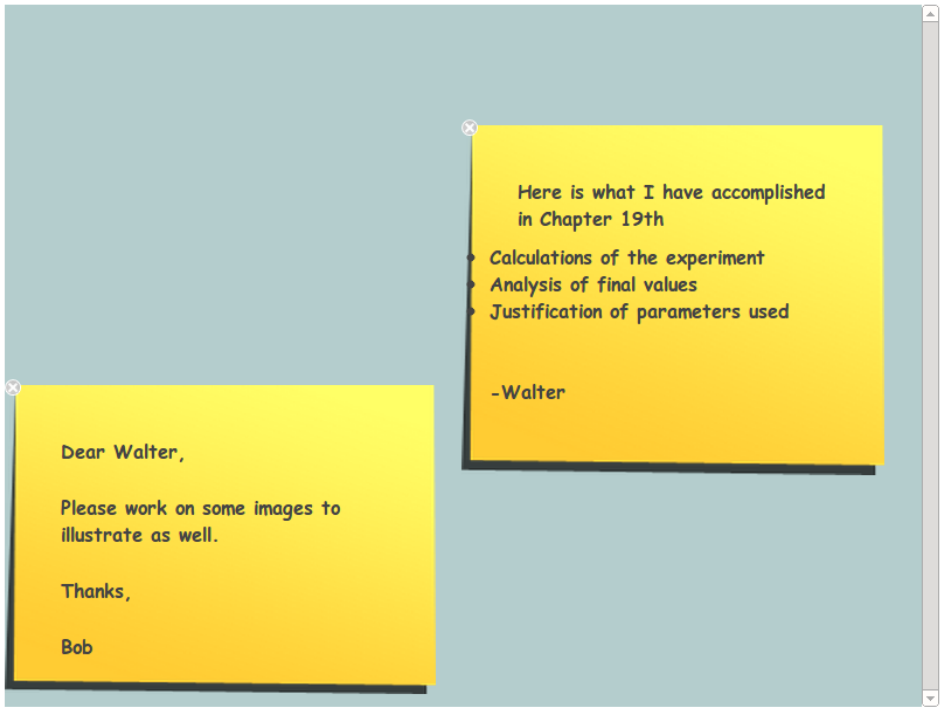

Figure 4.7: Sticky Notes Interface in Task Details page

The added note defaults to the top-left position of the board on which all notes are pinned. Post the note being created, the user can change the location of the text or size of the text. These edits are remembered and the next time the user comes to the same Task page, all the sticky notes take on the last modified position and size on the board. This functionality was added in order to enable the user to create non-overlapping notes positioned according to their preference. Thus every time before the page gets posted back to the server, a Javascript function gets called which:

- saves into a hidden html object the text, size, x coordinate and y coordinate of each new or edited sticky note seperated by a delimiter.
- retrieves all these parameters per note to be stored into a new Entity object in the Google Datastore.

Dropbox API Implementation

There are 2 features of Dropbox that have been added into BusyBee-TaskManager website, both of which are in the Task Details page. They are:-

1. Dropbox Chooser
2. Dropbox Uploader

1. Dropbox Chooser Implementation

   As mentioned earlier, Dropbox Chooser allows users to return web links to dropbox files from their accounts. This feature is useful for BusyBee-TaskManager since users can support their task completion status facts with documents detailing the work they have accomplished.

   First the user will have to click on 'Choose from Dropbox' button, which will open a Dropbox user Sign In interface (Figure 4.8). Signing in, will in turn open up a File Viewer Interface (Figure 4.9) from where the user can select their file and click on 'Choose'. This will cause the file link to get appended into the TinyMCE text editor (Figure 4.6)

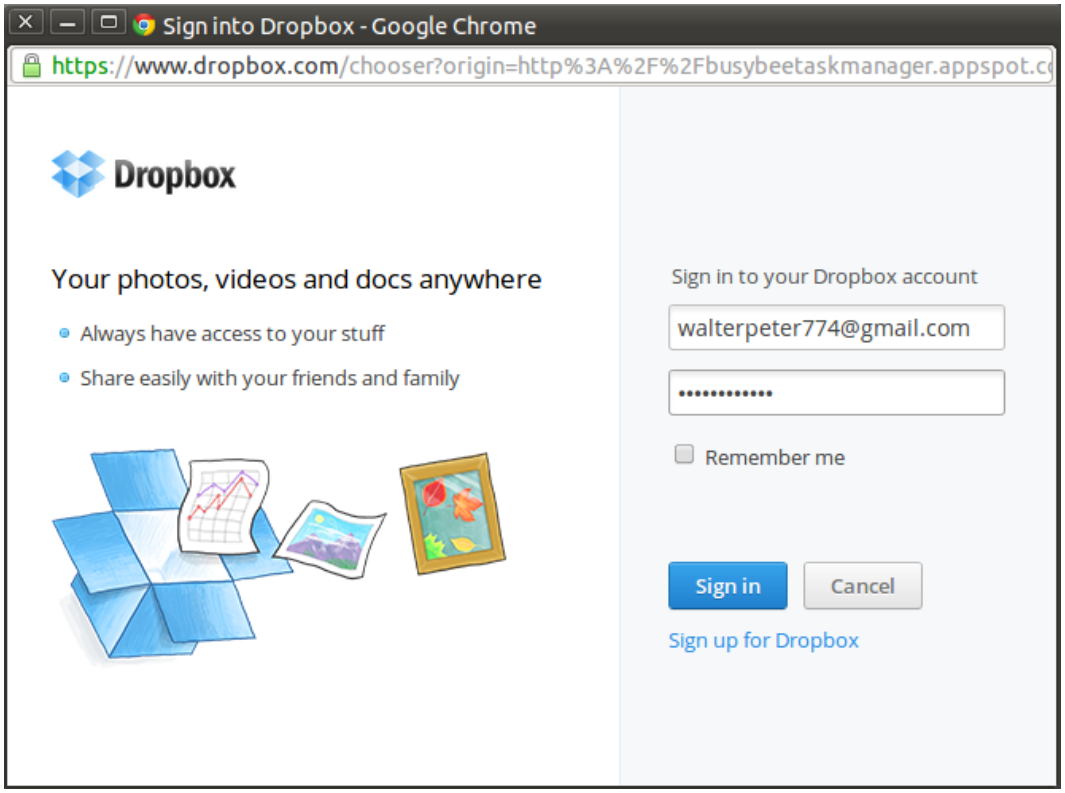

Figure 4.8: Dropbox Sign In Interface

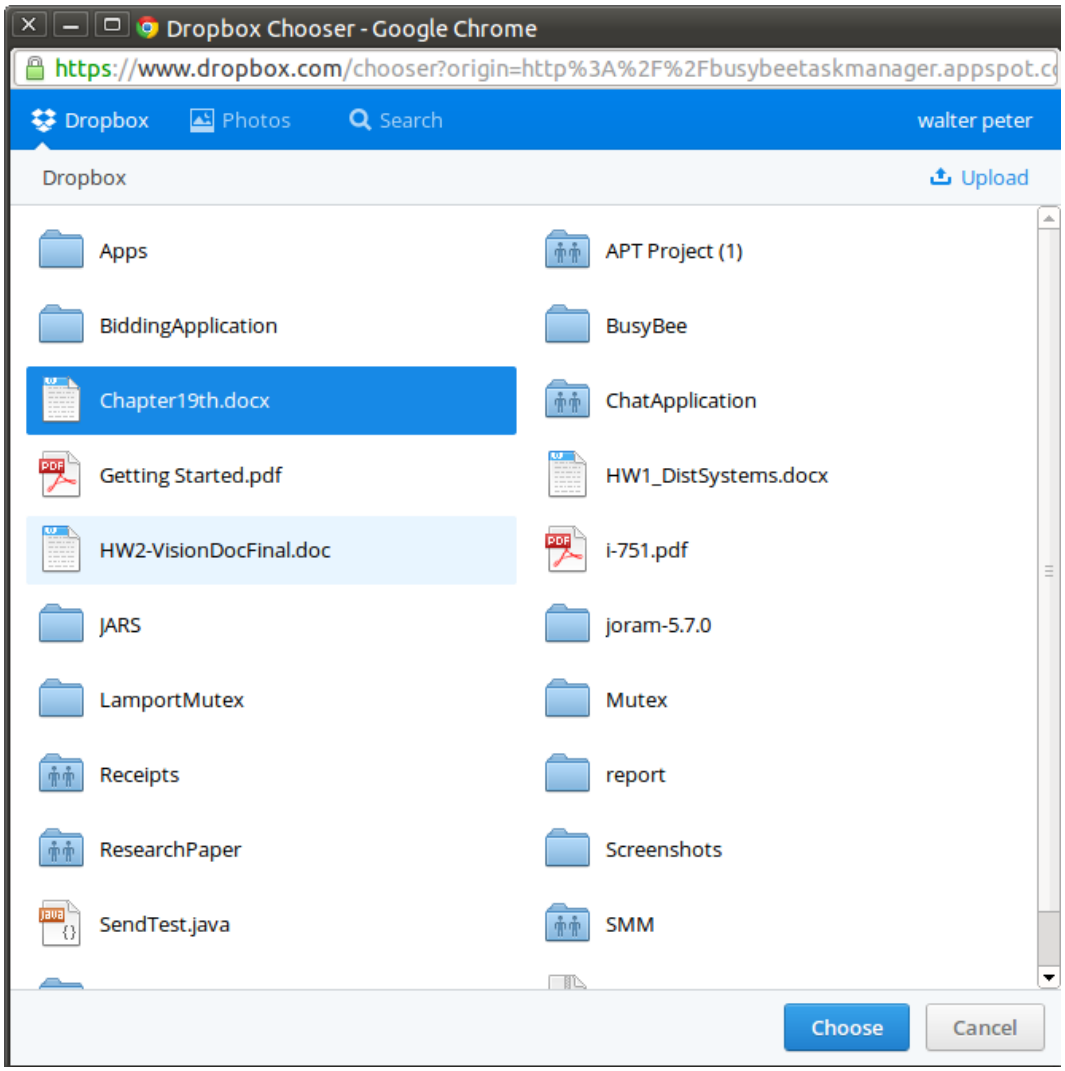

Figure 4.9: Dropbox Chooser Interface

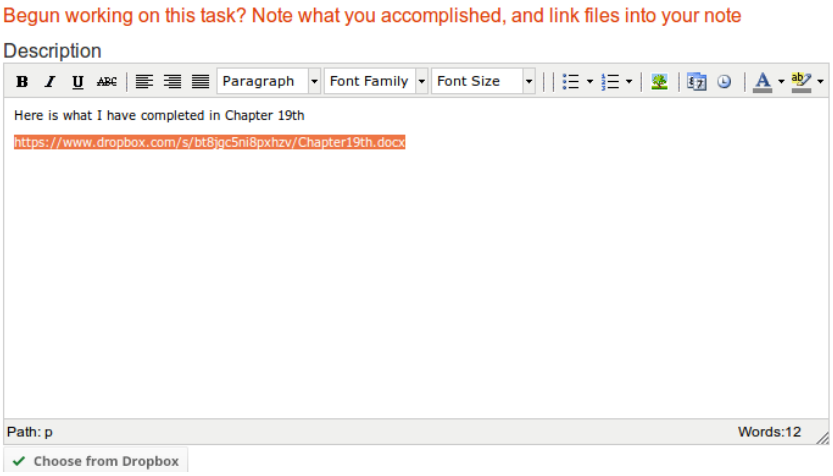

Figure 4.10: Dropbox Chooser returns web-link of file

I will now talk about the implementation of Dropbox Chooser. In order to implement the Dropbox chooser functionality, one has to do the following:-

- Register the application with Dropbox. On Registration, the dropbox developer gets an application key.
- Use this application key in the script source tag, where the Dropbox Javascript file is included.
- Add an event listener to listen to the event of a file being chosen by the user successfully. This would then be processed to obtain the web-link.
- Obtain and append the web-link object in html format to the text in the text editor.

The Dropbox Chooser Javascript library has a function to take care of the user Sign In, so that is something that the developer need not handle in the implementation.

2. Dropbox Upload Implementation

   I thought it would be useful to allow users of BusyBeeTaskManager to upload files on the go, through my website, as a natural sequence of task completion, as opposed to handling file upload at a later time. It is comparatively easier (in terms of fewer clicks) to open different types of files in Dropbox. Several other applications like Google Drive do not render uploaded files in the same way that they were created and prefer the document to have been created online. Google Drive in particular at this point in time, has trouble exporting embedded images in documents. The key element on the user interface side is the HTML5 File Upload Control. There are 2 ways to upload a file through this control. The first is through a file selector mechanism, which opens a file explorer interface from which the document can be selected. This is achieved by adding a standard file input placeholder. The second way is through drag and drop of the file into the File Area.

   On the Html front- end, there is a *div* element which is tied to a Javascript file to do some client-side computation based on user actions performed on the interface. In this Javascript file, there are listeners for events, *hover* and *select*. I referenced a tutorial on Html5 File Upload[11] to

build this feature. In addition, I wanted to obtain the parsed file and upload it into Dropbox. Therefore in a seperate Javascript function called *ParseFile()*, I obtained the file object and attached it to a hidden input element, which is later processed on the server-side on post-back (which happens post the user *dropping* the file onto the File Area or choosing the file from the selector).

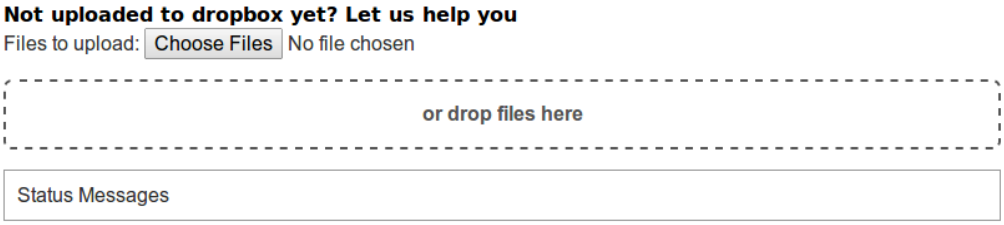

Figure 4.11: Dropbox File Upload Interface

Yet again, the developer has to register the application for a second time in order to use the Upload API. This is because in case of Upload, along with the *App Key*, the developers need to obtain an *App Secret* from the registration of their application, in order to put these components into the code. So I registered my application under a second name in Dropbox to obtain the 2 components. Let us now go over the functionality embedded on the Java Servlet side for the file upload.

- Use the *App Key* and *App Secret* to get the user to permit the website to access his or her DropBox account. This is a one- time authentication and I have stored the access token generated by Drop-

box into the user entity in the Google Data store, so that the user does not have to re-authenticate for uploading next time.

- Obtain the URL to which the user has to be redirected, in order to Sign In to Dropbox. On a successful Sign In, an access token is generated.
- Store the access token into a field of the user entity, in the Google Data Store.
- Create a Dropbox API object, after referencing the Dropbox upload libraries. Obtain the file object from the front-end and upload this file by serializing into a byte stream and using the *putFile()* method.

Disqus Implementation

Disqus implementation was a simple 2 step process. The first step was registration, in which I had to register my website to the Disqus platform. The second step was adding the Disqus script (which can be picked up from their documentation), into the page in which I was implementing Disqus. However in the second step, the developer needs to remember to modify the configuration variables as per need. Now, some websites need their discussion forum at a static URL and the query parameters should not change the forum location. However my website needed a seperate discussion forum based on the combination of task and project variables, so I did not have to modify the target URL parameter to stay constant.

Figure 4.12 shows the screenshot of the web page, post implementing

the Disqus platform. It can be seen that a comment has been posted by me, for a specific task. I can post with my username as I am signed in, in the screenshot shown in Figure 4.12 and therefore, it recognizes me. Otherwise Disqus would require me to Sign In to the account using which I was registered into Disqus (it could be Twitter, Facebook, Google or Disqus), to post the comment. If they are a new user, they can register into Disqus with either of the above mentioned accounts. The Sign In tab appears right below the Text Area for the comment.

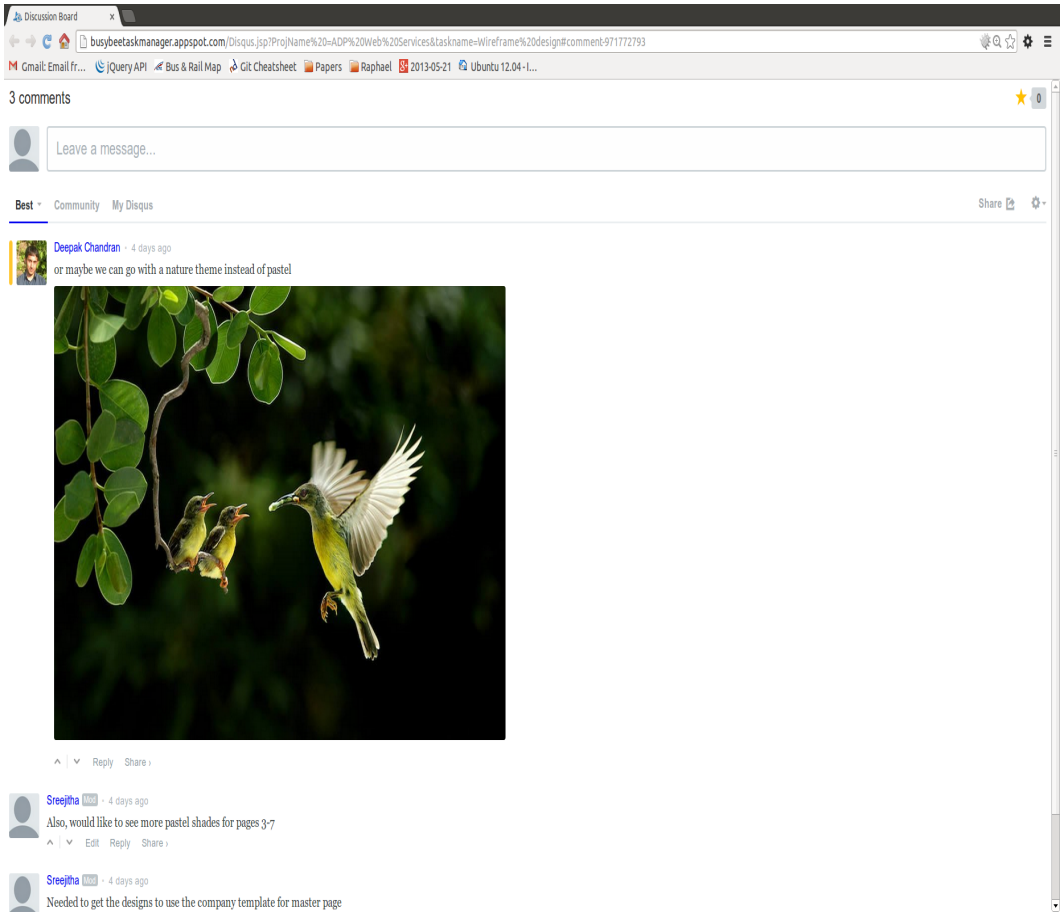

Figure 4.12: Disqus implementation on BusyBee website

Send Hub SMS Service Implementation

Send Hub, as mentioned before, is a service which enables people or applications to send messages to mobile phones. On task completion, dependent tasks of the current task are obtained. Let us assume that Task B is the only dependent of Task A. Therefore when Task A is completed, the list of dependent tasks (In this case only Task B) that are clear for start are ob-

tained. The cellphone numbers of the task owners of these tasks are looked up and the message is sent out. Thus part of the implementation of SendHub is in the registration page of BusyBeeTaskManager. When a new user registers into the website, his phone-number is used to create a new SendHub contact. A JSON formatted Http request is sent to the SendHub site. I have parsed the response received using regular expressions to obtain the id assigned to this newly registered user. This id is then stored as a field of the user entity in Google Data store. The other part of the implementation lies in the Java Servlet redirected to, on submit of Task Details page. This is much similar to the format of adding a SendHub contact, with a few changes. Here I have first obtained the SendHub id of the user, from Google Data Store (so the steps are somewhat of a reverse of adding a contact). I then implemented a function which again sends a JSON formatted Http request to SendHub, but this time it is a request to send a message, instead of adding a contact. The type of request is differentiated by the JSON parameters that are passed. Thus if the parameters passed are 'contact' and 'text' then the request is to send a message to a SendHub contact whose id is the value of the key 'contact'.

4.3 Home Page

As required by the specifications, I designed my Home Page, to pictorially convey what the website provides and to have links to the View Projects page if the user is already registered. However if the user is not registered, then instead of the View Projects page, this page can link the user to the

Registration page.

Figure 4.13: Home Page. The purpose of the home page is to convey the simplicity of BusyBeeTaskManager so that the user feels welcome by it.

Whether a user is new or returning is determined with the help of a simple cookie which posts back the user's gmail address to the servlet. This address is checked against all currently registered users to determine whether it already exists in the data store. Figure 4.11 depicts a screenshot of this page post implementation.

4.4 Registration Page

I designed a basic form to record user details, of which, phone number and photograph are the most important (But all fields are required). All these

details are captured into an entity called *User Details* in Google Datastore.

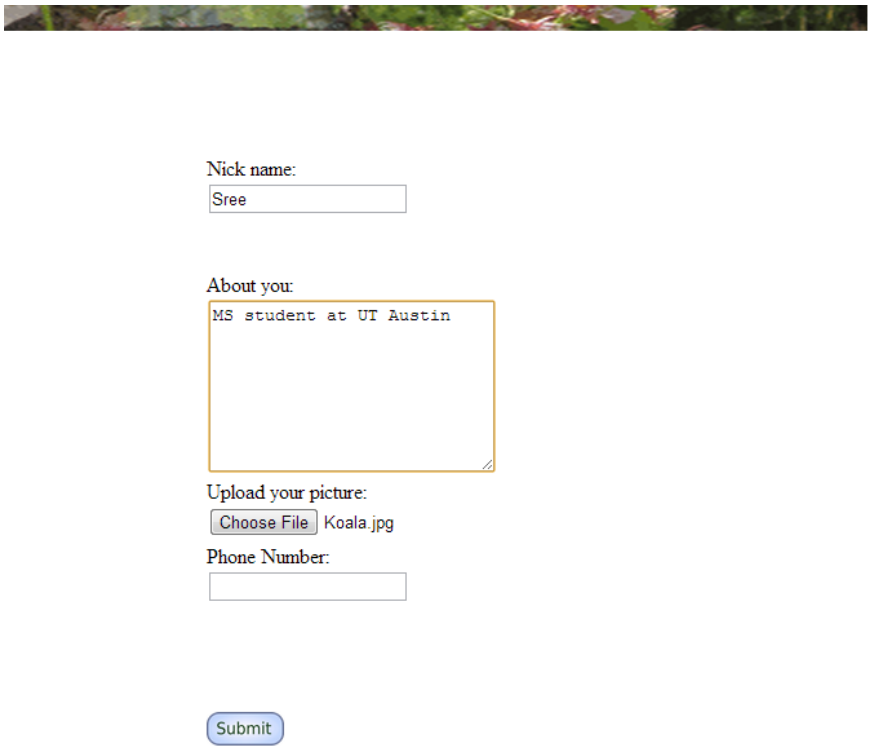

Figure 4.14: Registration Page. Registration is needed for BusyBeeTaskManager to automatically send SMS and email notifications as well as show profile pictures of group members working on a project.

The phone number and email are used to subscribe to Task Notifications. The photograph is used for display on the *Task Details* page of those tasks for which the user is the owner. The photograph is stored as a BLOB object into Google Blobstore and is rendered on pages which request it, via the Google Image Service. Figure 4.12 depicts a screenshot of this page post implementation

4.5 Start Project Page

I designed a basic form to facilitate the user to start a project. A key feature is that when the user selects a team member, their pictures appear in a vertical grid across the right hand end of the web page. I implemented this, using Google Image Service. During registration, the submitted photograph as mentioned earlier is fetched into Google Blobstore. Google Blobstore annotates each of the blob objects with a unique key. I retrieved this unique key, post insertion of the blob object into the store and re-inserted it as one of the properties of the entity *User Details*.

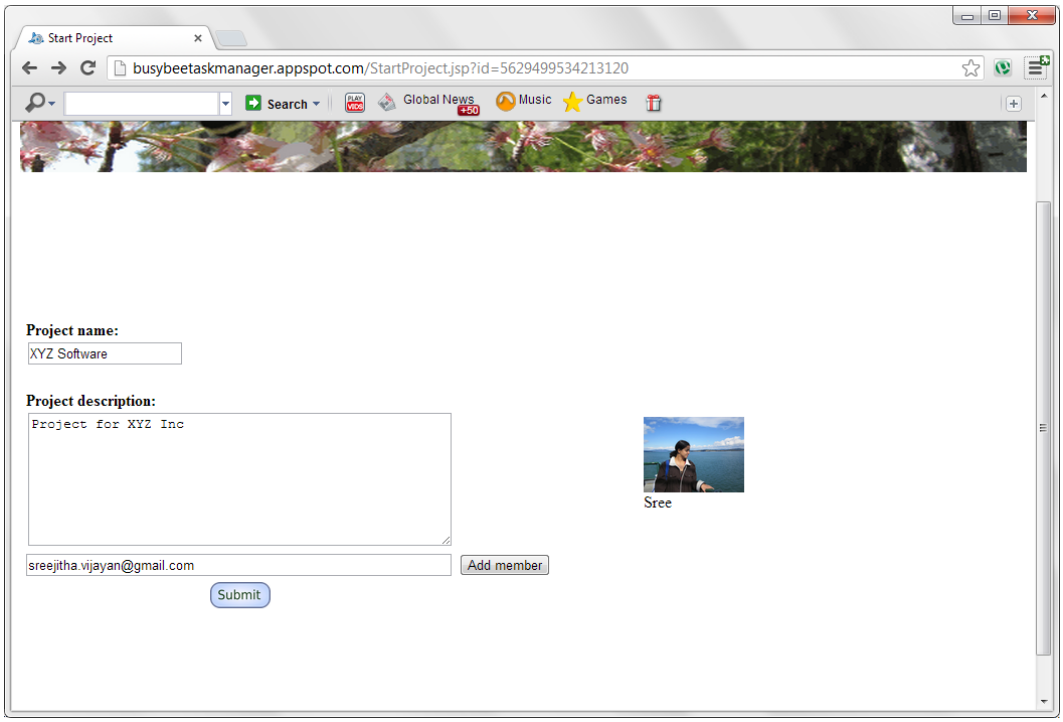

Figure 4.15: Start Project Page. The start project page allows a user to describe a new project and invite members to join the project.

Thus when a user is selected to join the team by another user in the *Start Project* page, the image is retrieved from the Blobstore using this key, which is in turn retrieved using the selected user's email address. Figure 4.15 depicts a screenshot of this page post implementation.

4.6 View Projects Page

This page, as required, displays to the user all the projects that he/she is a part of, in a grid format. Also, as mentioned before, I used the Slickgrid API to render the grid. I needed to render the Project Name column as a hyperlink to the Task Manager page. SlickGrid requires manual specification of the data format of each column in the table, in order to determine its representation. So for the project name column, I specified the format of the data as a hyperlink. The hyperlink upto the page name which is *Task Manager* is constant, only the parameter for project name had to be added into the name placeholder. Thus for two projects whose names are 1 and 2 respectively, the static part of the hyperlink would be as follows:-

https://busybeetaskmanager.appspot.com/taskmanager?Projectname=

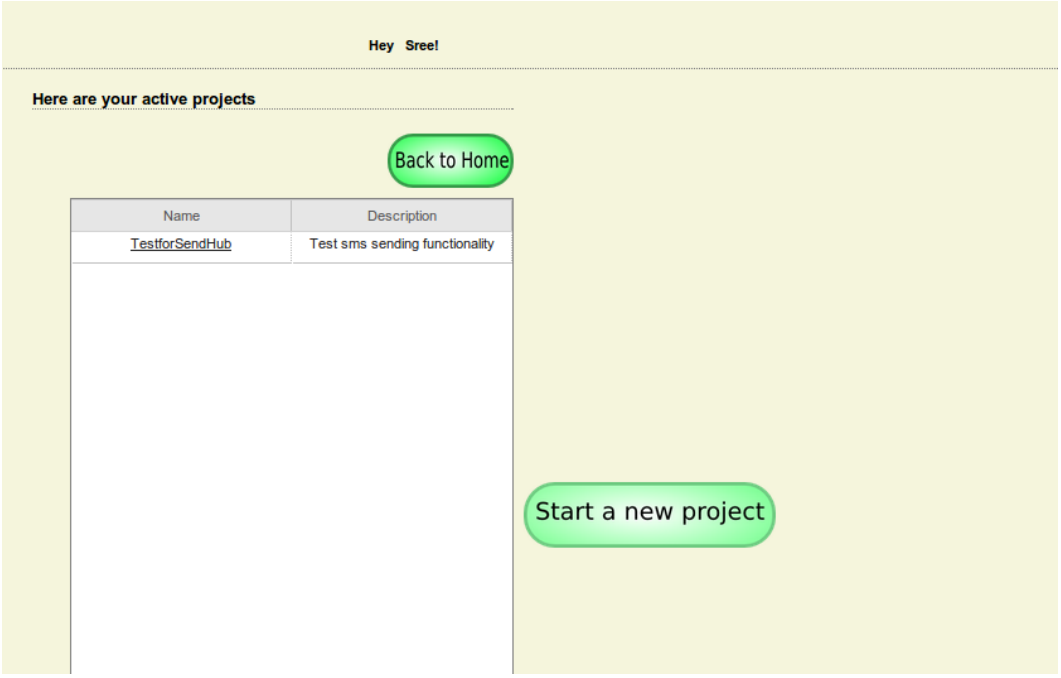

Figure 4.16: View Projects Page. The purpose of the View Project page is to provide the user with a central portal to all of his or her projects.

The value for 'Projectname' is fetched in json format, from the Datastore. The project description is also fetched from the Datastore for the second column of the grid in the same way. Next, this fetched value is appended into the static hyperlink after the '=' sign. The format specification, ensures that the column is rendered at page load as a hyperlink. Figure 4.16 depicts the *View Projects* page in the website post implementation.

Chapter 5

Results

Measurement is crucial for sound evaluation and continued improvement of any software project. Hence, this chapter will showcase results in terms of Software Metrics (Complexity and Size), Application Performance and walk-through of the website with Demo Videos.

5.1 Software Metrics

Since this project began initially as part of a class project and later expanded into a thesis report I thought it would be interesting to map the development trends and metrics over time. I used a program called SVNPlot [17] to generate the plots from my SVN repository's logs. SVNPlot is a python program that uses SQLite3 to convert SVN's logs into a database. It then analyzes the database and generates plots (PNG files) using Python's Matplotlib module. The instructions to use this program are available in the site's wiki page. For SVNPlot I looked at my commit activity, average and total lines of code and file types.

5.1.1 Code Size

I only ran queries which were relevant to understanding code size on the SQLite3 database. There are many more interesting results that can be derived with SVNPLot such as contribution in terms of author, days of the week when more commits took place etc. From the results of running these queries (plotted in Figures 5.1 to 5.5), I observed that the total lines of code are between 7500 and 8000. However, a major contribution to this number comes from the Javascript libraries that were reused, which lie in the 'war' folder. Also, it can be seen that commits were frequent during Dec 2012 and May 2013, which is agreeable with, as I was working towards the semester end and report deadlines during these times.

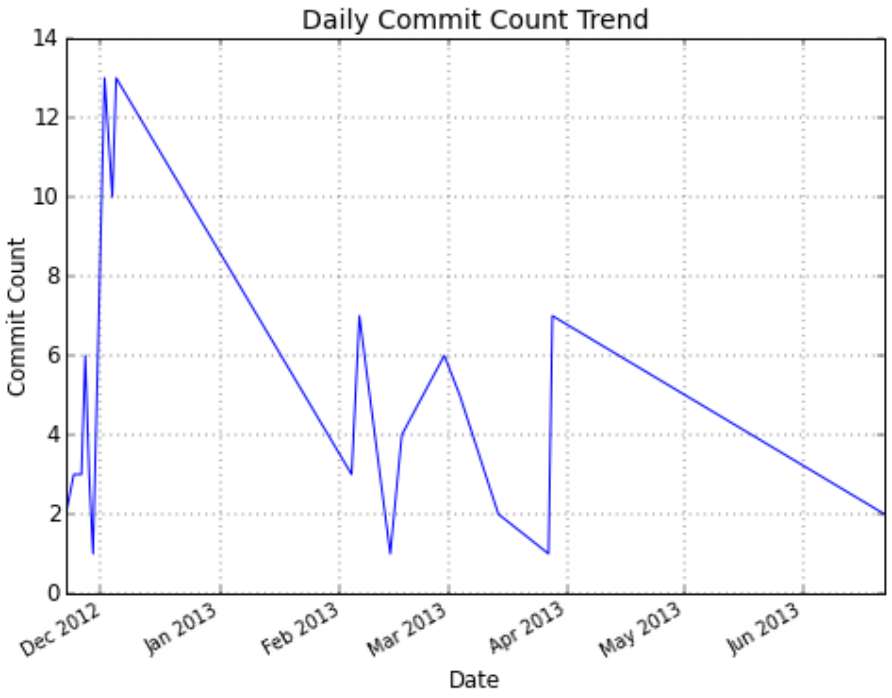

Figure 5.1: Number of commits per day during the last several months. From this plot it is fairly clear that most of the coding work occurred during the months of December and January. After that, there was some increase in activity during May, but much less than December

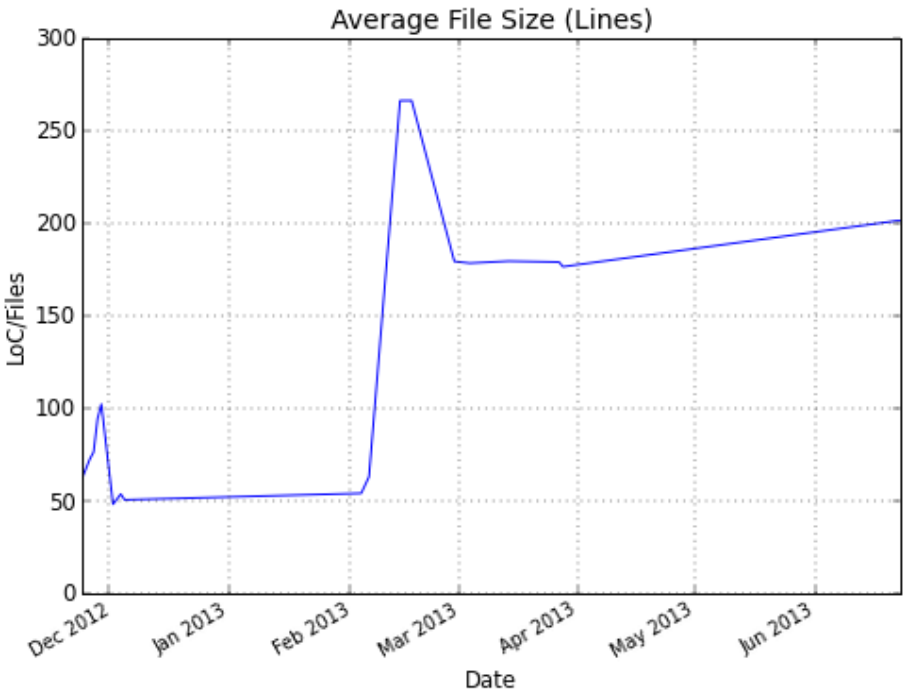

Figure 5.2: Average lines of code per file. The trend here shows that there was a push toward modular code. Whenever the lines of code per file spiked (increased), there was an immediate response to bring the file size down by splitting the code between multiple files.

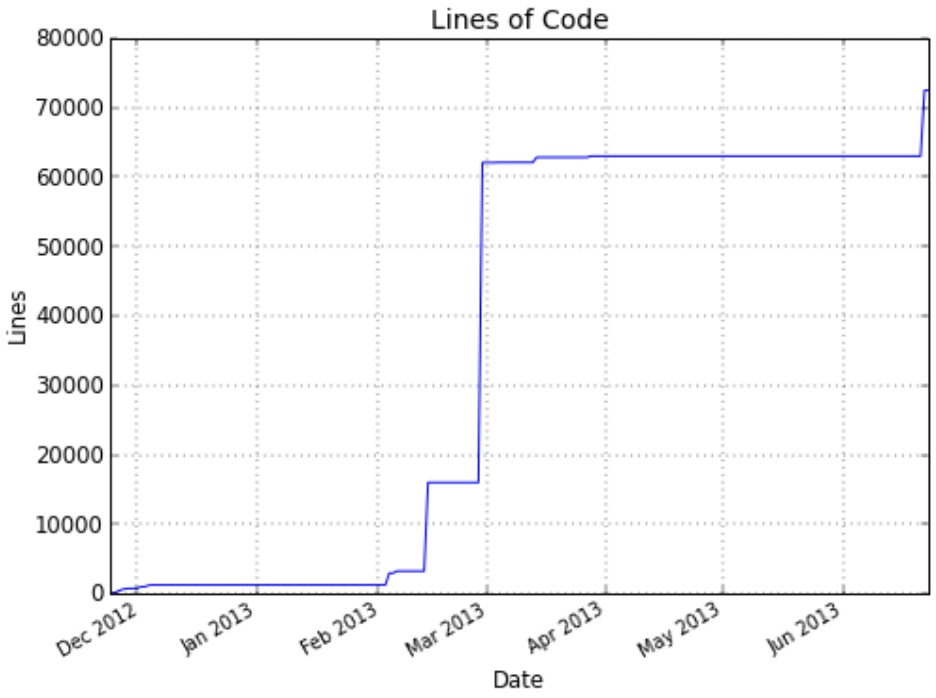

Figure 5.3: Total lines of code. Many lines of code were contributed by large libraries and therefore, sudden unreal increase in the lines of code indicate that some third-party library was incorporated into the project. However, along with third-party libraries, a lot of other code was also changed. Therefore, these sudden changes are really a combination.

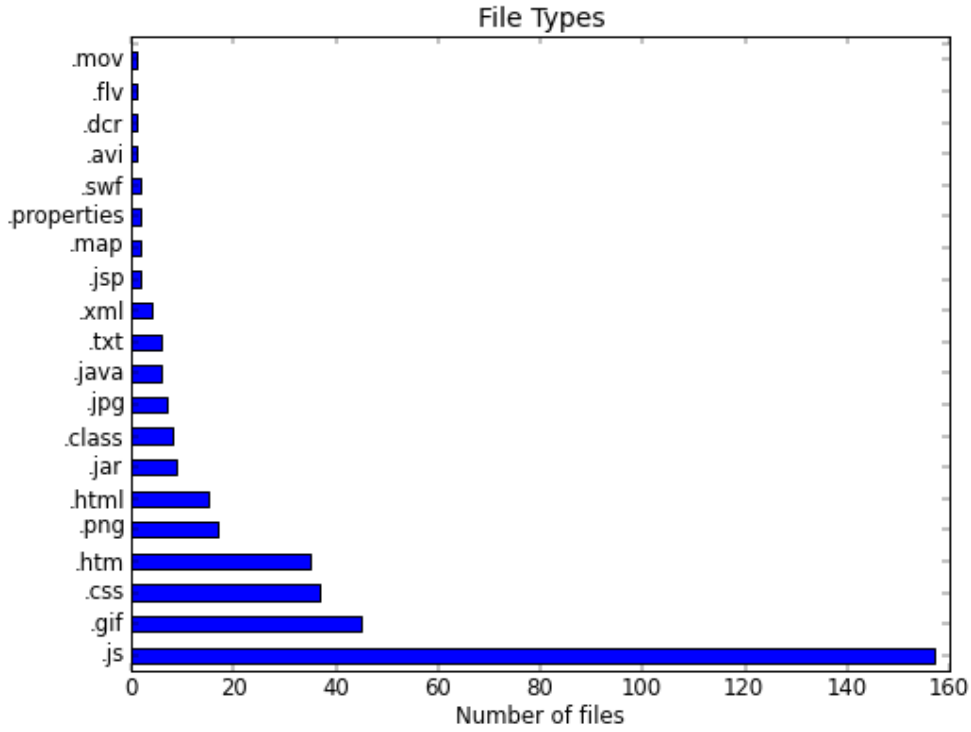

Figure 5.4: Number of files of each file type. Javascript files largely come from libraries that were reused, such as RaphaelJS.

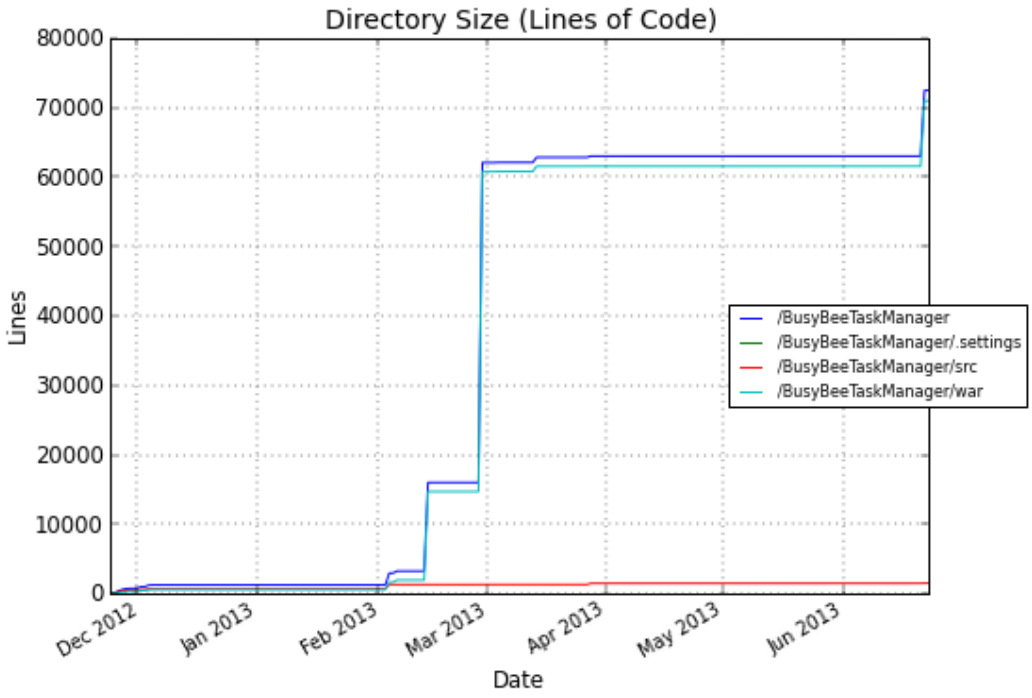

Figure 5.5: Code size within each directory. The trend here suggests that the war folder was the main contributor to the lines of code.

5.1.2 Web page loading time

It was imperative to understand website performance in terms of speed of rendering pages. This was measured using Chrome Dev tools, available with the Chrome browser. It was known beforehand that the maximum page load times would be taken up by the Task Details and Task Manager pages, attributed to the number of components in it. And sure enough, Task Details page takes the maximum time with 7.9 seconds in the test scenario as shown in Table 5.1.

Since 7.9 seconds was higher than expected, I was curious to examine it in detail. I found out that all the visible components loaded completely in 300 milliseconds. But several TinyMCE Javascript library files, loading in the background (invisible to the page), took up around 7.9 seconds. And page load time, is the maximum time taken by any component to load, which in this case were the TinyMCE Javascript library files. Thus bringing down the load time of this page, qualifies as a future work item.

Web page	**Loading time**
Starting Page	521 milliseconds
View Projects Page (14 projects)	1.8 seconds
Project Details Page (12 tasks)	2.3 sesonds
Task Details Page	7.9 seconds

Table 5.1: Loading Times for the different websites in the project (measured using Chrome Dev Tools)

5.1.3 Code complexity

Looking at code complexity is important for understanding areas in the code that might need to be refactored in order to increase maintainability and modularity. I used different tools to measure the code complexity of different source types, primarily Javascript and Java. JSP code is not analyzed because I was unable to find a tool that would measure complexity of JSP files, partly because the JSP files in BusyBeeTaskManager contain a combination of Javascript, Java and HTML code.

Javascript code

A Javascript-based webapp called JSComplexity[1] was used to measure the code complexity for the TreeLayout.js source file, which is the longest Javascript source file used in BusyBeeTaskManager. The TreeLayout.js code is responsible for rendering the tasks on the Project Page. From the output below, the most important value is the Maintainability index, which is relatively high for TreeLayout.js. This indicates that the code is relatively easy to maintain and therefore, complexity is manageable. The average cyclomatic complexity is low, but the complexity of one of the functions, *draw()* is fairly high. The *draw()* function is indeed slightly confusing and therefore, this analysis correctly pointed at the function that needs some refactoring.

Measure	Value
Maintainability index	90.70095974982883
Logical LOC	155
Average cyclomatic complexity	5.20

Table 5.2: Code Complexity- Javascript Code

Function Name	Logical LOC	Cyclomatic Complexity
TreeLayout	9	1

Table 5.3: Code Complexity for TreeLayout - Javascript Code

Java Servlet code

I used a program called JavaNCSS[12] to calculate the code complexity for Java-based source files. JavaNCSS is a command-line based tool that outputs cyclomatic complexity for each function as well as the average. It is available for download as a bat file. At a very basic level, non commenting source statements are recorded by this program. Tables 6.4 gives the average complexity values for the Java Code, while Table 6.5 gives these values at the function level of Java code.

Function Name	Logical LOC	Cyclomatic Complexity
TreeLayout.addNode	1	1
TreeLayout.removeNode	9	1
TreeLayout.updateConnections	30	6
TreeLayout.setMouseClickEvent	2	2
TreeLayout.draw	71	14
TreeLayout.nodeselected	10	1
TreeLayout.nodedeselected	10	1
TreeLayout.calcDepth	12	6

Table 5.4: Code Complexity breakup for TreeLayout - Javascript Code

Avg Function Non Commenting Source Statements	Avg cyclomatic complexity	Program Non Commenting Source Statements
21.70	5.20	596.00

Table 5.5: Code Complexity for Java Code - JavaNCSS

Sr. No	NCSS	CCN	Function Name
1	9	4	busybee.AddProjectServlet.stringJoin
2	16	4	busybee.AddProjectServlet.createNewProject
3	7	1	busybee.AddProjectServlet.doPost
4	32	9	busybee.AddTasksServlet.createTasks
5	39	9	busybee.AddTasksServlet.printTasks
6	22	6	busybee.AddTasksServlet.modifyTask
7	31	11	busybee.AddTasksServlet.removeTask
8	13	7	busybee.AddTasksServlet.sendSMS
9	12	3	busybee.AddTasksServlet.sendEmail
10	3	1	busybee.Dropbox.doGet
11	16	2	busybee.DropboxUpload.doGet
12	23	1	busybee.RegistrationServlet.doPost
13	6	1	busybee.SendHubAddContact.AddContact
14	35	7	busybee.SendHubAddContact.excutePost
15	5	1	busybee.SendHubSMS.SendSMS
16	26	4	busybee.SendHubSMS.excutePost
17	100	14	busybee.TaskManagerServlet.generateJavaScript
19	4	1	busybee.TaskManagerServlet.doPost
20	2	1	busybee.TaskManagerServlet.doGet

Table 5.6: Code Complexity for individual functions in Java Code - JavaNCSS

5.2 Link to video

- **User Registration and Project Initiation**

  The following video link walks through the process of user registration and beginning a new project by adding team members and tasks. It also demonstrates the creation of the Task graph based on rows entered into the Task table.

  http://www.youtube.com/watch?v=KsDrqnv6RhI

- **Dropbox Upload and Task Update**

  The following video link demonstrates how a user can edit task status-add notes, completion status and deadline date. It also shows how users can upload files into their dropbox account and link these files into the notes.

  http://www.youtube.com/watch?v=1fzvKjgFqfQ

- **Disqus Commenting System**

  The following video link demonstrates how a user can participate with task owners and other team members, each of these discussion boards unique to each task of the project.

  http://www.youtube.com/watch?v=32Ur9PHG9wc

Chapter 6

Conclusions

6.1 Summary

All the specifications and goals set out in the beginning of the project have been met. Following is an overall summary of features and functionality of the implemented system.

1. BusyBeeTaskManager is a Web App in order to make collaborations easier. (no need to download)

2. BusyBeeTaskManager supports cloud storage for further support of collaborative development.

3. BusyBeeTaskManager shows task dependencies visually in order to provide a quick overview of a project.

4. BusyBeeTaskManager enables sharing and uploading files with Dropbox.

5. BusyBeeTaskManager's GUI is designed to be friendly and non-intimidating so that a variety of users can use it for a variety of projects.

6.2 Conclusions

1. Lot of newer APIs do not support your favorite programming language. Hence be armed and ready to interpret code in another language or implement functionality anew.

   I experienced this issue while implementing the SMS Service and File upload Functionality. A popular choice for SMS Service was Twilio, which was supported only in Python, hence I could not use it. Even for implementing SendHub SMS Service, I had to translate code written in cURL script, to Java Http Post request, which took a lot of time. As for Dropbox, although Dropbox Java API can be downloaded, clicking on the link would lead me to the wiki pages for Dropbox Android API, whose implementation was not very similar. So implementing the Dropbox file upload functionality was a trial and error process all the way.

2. Be careful with extremely concise tutorials

   For the Sticky Notes Functionality, I initially referred to the documentation in the jQuery website. It was a one-pager with no explanation as to what purpose certain lines of code served. Despite following it word to word, I ended up with a blank *notes wall*, without an event firing on the click of 'Submit' button. I was fortunate to have come across a StackOverflow page, which said that a stylesheet was not mentioned in that tutorial, which caused all these problems. I had further issues, when I was using another jQuery GUI component (Date Picker) in the same

page, which also came with its own style sheet. These two style-sheets contradicted each other in styling effects, causing problems in the calendar display. I had to tweak a section of the calendar style sheet to get it to work properly. Now, when I come across extremely concise tutorials, I definitely double-check on them with popular developer forums.

3. SVN misplaces the class files in the repository.

   This story might just encourage to follow check-in best practices. I was lazy to keep track of new files in my folders, so instead of adding the new files one by one as and when they were created, I would do an equivalent of adding all new files much later from the root directory using:

   ```
   svn add *.*
   ```

   It would warn me of already versioned files and I would skip adding those. But for some reason, it ended up adding all my class files inside the folder where only my java files were to reside. And I would spend extra time in cleanup. I realized my folly and began committing as soon as a new file was added, to save the trouble of remembering its name after significant time had passed. So regular commits are extremely necessary and time efficient.

4. Subscribe into the news feeds of software products and organizations you have come to admire. Read their developer blogs often.

I love Dropbox as a backup tool and had subscribed to its weekly news digest. I got my Dropbox upload feature idea from one of the blogs which introduced their latest offering in the Dropbox API and decided to try it. It fit in well with my application needs and I was happy that I had invested time into reading their periodic blogs. Thus, it helps to be upto date with the progress and revisions that applications you admire are making, because these just might be a huge contributor to your next software project.

6.3 Related Work

Let us now compare applications, which similar to BusyBeeTaskManager are tools designed for project and team management.

Lombardi Websphere

Lombardi[10] (Figure 6.1) is an on-premise tool by IBM for Business Process Management. It was built with the goal of helping enterprises improve their productivity matrix. Lombardi's authoring environment is an Eclipse based tool and written in Java. Lombardi Websphere developers, can use this tool to author workflows and integration services of their tasks. A big advantage is that each developer can maintain their individual authoring environments and all their artifacts get versioned into the Lombardi servers. Lombardi also comes with a process console to manage all the artifacts. A key disadvantage is that Lombardi is resource intensive and requires multiple web

servers and databases pre-configured on the machine in order to be installed. Also, it saves every commit into a new screenshot, which depletes a lot of memory. Lombardi is an enterprise-level tool and might be a bit intimidating to use for a novice programmer or a non-programmer. It is also not cost-effective for small scale projects.

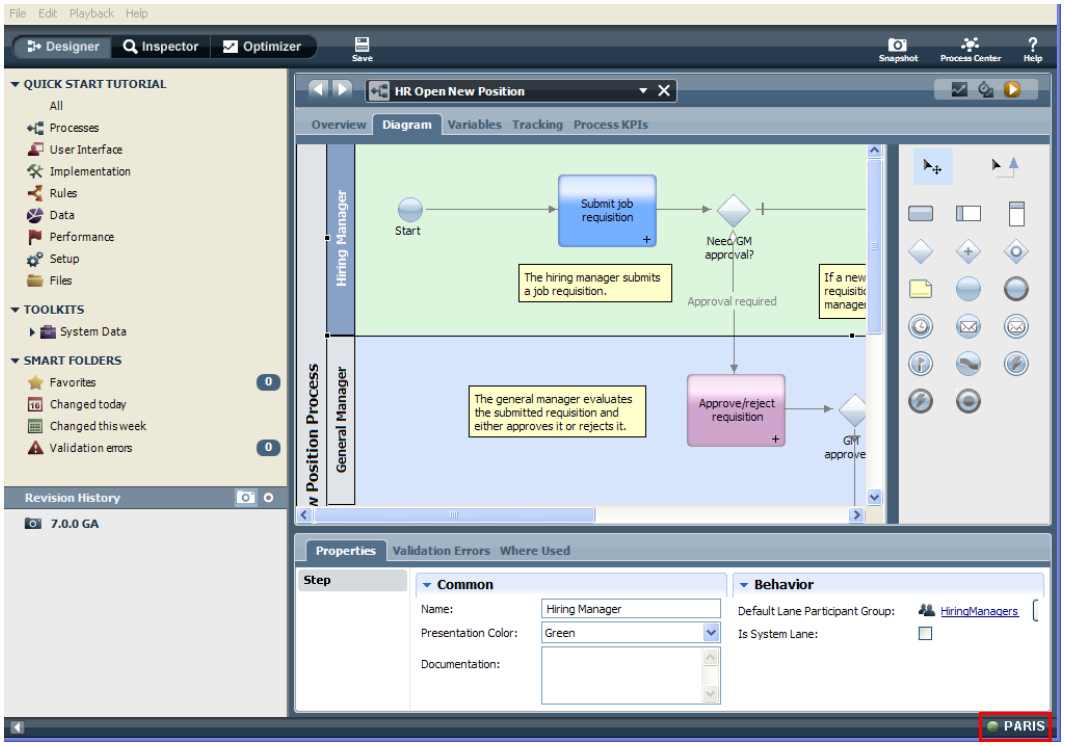

Figure 6.1: Lombardi Websphere process manager

Asana

Asana[3] (Figure 6.2) is a task management tool that supports categorization of tasks, upload and download of files via Dropbox, discussion boards and alerts. Unlike Lombardi, Asana provides a fairly friendly UI, which is us-

able by a non-technical user. Asana supports mobile devices: Android phones, iPhones and iPads. The wide range of support for mobile platforms is where Asana has the main advantage over BusyBee. However, unlike BusyBee, Asana does not provide relationships between tasks. The alerts are directly sent by other users, whereas in BusyBeeTaskManager, the alerts are automatically sent using the dependency relationships of a task. Asana has support for deadlines, which was a planned feature for BusyBeeTaskManager that has not been implemented yet.

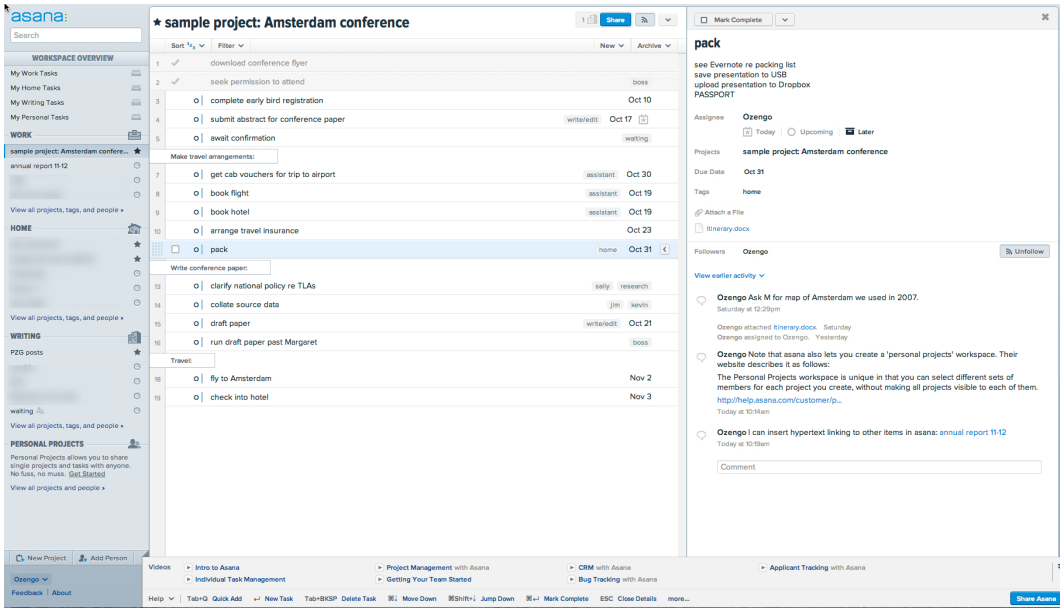

Figure 6.2: Asana task manager

Any.do

Any.do[2] (Figure 6.3) is one of the many rising 'to-do list'applications, particularly for mobile devices. Any.do is extremely simple to use, which is

one of the main reasons for its popularity. Like Asana, it provides a list of tasks that can be categorized. There is no support for cloud storage of files, discussion boards, or alerts. Any.do and similar planners are designed for simpler use cases such as grocery lists or household chores. But, even in simple situations, dependencies can exist. For example, cooking might require grocery shopping as a prerequisite. Any.do, however does not account for task dependencies.

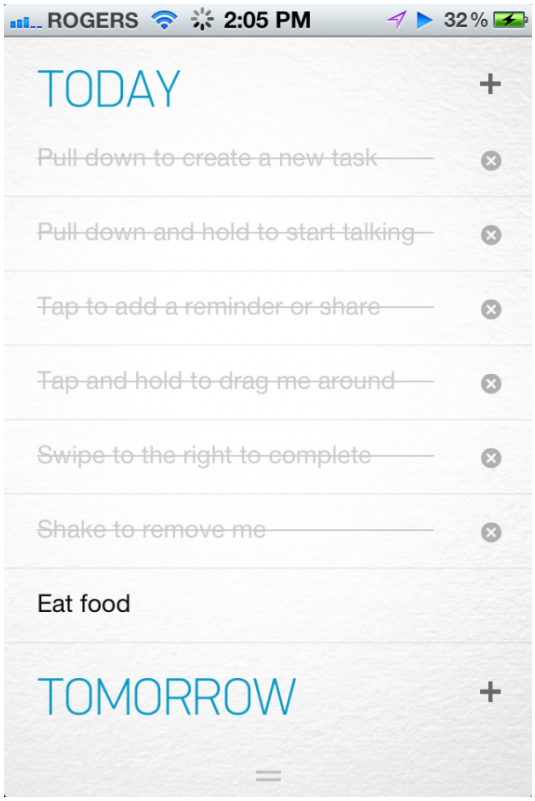

Figure 6.3: Any.Do task manager

6.4 Future Work

During the course of implementation of BusyBeeTaskManager, I had several ideas to make the website better in alignment with the latest trends in technology or to add more capabilities such as a recommender component. These ideas could not be implemented due to lack of time, however I would like to work on them in future. Some of these are elaborated in the next few sections.

Finding team members- Recommender

This feature would allow any BusyBeeTaskManager user to find suitable team members by specifying the skill-sets that the project needs. For example, suppose a software project requires a team member who knows how to program using the Python programming language as well as understands matrix algebra and computer graphics; BusyBeeTaskManager will be able to find the suitable candidates for this role.

The search problem described above can be implemented using a nearest-neighbor method. The search query is a vector indicating the required skills for the new team member. Each user in BusyBeeTaskManager will be represented as a vector of size N, where N is the total number of skills described in all BusyBeeTaskManager projects. Each user's vector will be populated using metadata from the tasks that he or she has worked on. For example, if a user has worked on three tasks that involved Python, then that user will have a score of 3 in the vector index corresponding to the skill 'Python'.

The nearest-neighbor method using vectors will be a fast and efficient way for BusyBeeTaskManager users to find other users who have matching skill sets.

Improved User Interface design

Some user interface (UI) improvements were considered during the course of development of BusyBeeTaskManager but left unfinished.

Currently, it is slightly inefficient for a user to quickly move in and out of different tasks. It would be ideal if the same web page shows both the task dependencies and the task details. This UI modification would involve adding some extra panes that would 'slide in' from the bottom, showing the details of the selected task. This way, a user does not lose sight of the task dependency graph when viewing the details of a single task.

Another user interface improvement, as noted in the previous chapter, would be shorter page load time for the Task Details page. So, an editor which provides the same capabilities as TinyMCE, but also shorter rendering time would have to be considered.

Deadlines and Time-Related Metadata

Most tasks in real projects have deadlines. Sometimes the deadlines are absolute, other times, relative. For example, the deadline for Task 1 might be "2 days before Task 2's deadline", in which case the deadline is considered relative. Both absolute and relative deadlines are necessary for making

time management automated in BusyBeeTaskManager. Thus if completion of a task is falling behind schedule, it would be color coded in a certain way. Another color code would apply if the task is complete, but behind schedule. Thus a combination of two variables, completion status and task plan, would determine the overall task status.

Android Push Notification

With smart phones becoming more commonplace, a shift is occurring in communication styles. App-based notifications have begun to replace generic modes of communication such as SMS and E-mails. This is partly because people can immediately classify the source and importance of the message in their current environment, based on the application that is providing the notification. For this reason, I would prefer to use push notification services for BusyBeeTaskManager's notifications instead of SMS notifications. (I would still maintain Gmail notifications). With SMS, users cannot immediately identify the source - they have to look at the notification provider in order to realize that it is a BusyBee message. Thus, if there was a dedicated App for receiving and storing BusyBee's messages, users would be able to receive the message more conveniently.

Payment Gateway

Currently, the website is free for use. But I am planning to come up with a cost model for registered users. The tentative plan is to provide

lifetime membership for every BusyBee user at $0.5 per month, which would be the basic subscription mode. For every 21st project or 301st task onwards, whichever comes earlier, it is planned that the pricing would change to $3 per month for the user. Thus, in order to conduct this task or project, the user would have to change the subscription mode from basic to premium. In lieu of this cost model, I am planning to add a Payment Gateway to the BusyBee website.

Bibliography

[1] Analysis - JSCcomplexity.org. http://jscomplexity.org/.

[2] Any.do: Good day. every day. http://www.any.do/.

[3] Asana: Task Management for teams. http://asana.com/.

[4] Build the power of Dropbox into your app. https://www.dropbox.com/developers.

[5] Disqus - The Web's Community of Communities. http://disqus.com/.

[6] EditableGrid, build powerful editable tables - What's EditableGrid? http://www.editablegrid.net/en/.

[7] Enterprise Social Networking and Collaboration Platform - Socialcast. http://socialcast.com/.

[8] Google App Engine - Google Developers. https://developers.google.com/appengine/.

[9] Google Cloud Messaging for Android - Android Developers. https://developer.android.com/google/gcm/index.html/.

[10] Help - IBM WebSphere Lombardi Edition Version 7.2 Information Center. http://pic.dhe.ibm.com/infocenter/wle/v7r2/index.jsp?topic=%2Fcom.ibm.wle.doc%2Frn%2Ftopic%2FWhats_new_in_TeamWorks_710_rn.html.

[11] How to Use HTML5 File Drag and Drop-SitePoint. http://www.sitepoint.com/html5-file-drag-and-drop/.

[12] JavaNCSS - A Source Measurement Suite for Java. http://www.kclee.de/clemens/java/javancss/.

[13] Notification Service-Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Notification_Service.

[14] Public Cloud Hosting, Computing, Storage and Networking by Rackspace. http://www.rackspace.com/cloud/.

[15] SendHub API 0.1 documentation. http://apidocs.sendhub.com/GettingStarted.html/.

[16] SlickGrid. http://nugetmusthaves.com/Package/SlickGrid.

[17] svnplot - Generate Graphs from Subversion log statistics. https://code.google.com/p/svnplot/.

[18] TinyMCE. http://www.tinymce.com/.

[19] Twilio Cloud Communications: APIs for Voice, VoIP and Text Messaging. http://www.twilio.com/.

[20] Visualization: Table-Google Charts-Google Developers. https://developers.google.com/chart/interactive/docs/gallery/table.

[21] Yammer: The Enterprise Social Network. https://www.yammer.com.

[22] jQuery, 2005. http://www.jquery.com.

[23] SlickGrid or FlexiGrid, 2013. http://stackoverflow.com/questions/15783316/slickgrid-or-flexigrid/15788263#15788263.

[24] Dmitry Baranovskiy. Raphael–javascript library, 2011.

[25] Ryan Barrett. Under the Covers of the Google App Engine Datastore. *Google I/O*, 2008.

[26] Amazon Elastic Compute Cloud. Amazon web services. *Retrieved November*, 9:2011, 2011.

[27] Costan. Some love for Javascript applications, 2012. https://tech.dropbox.com/2012/08/some-love-for-javascript-applications-2/.

[28] JQuery Foundation. *Jquery reference guide: a comprehensive exploration of the popular javascript library*. Packt Publishing Ltd, 2010.

[29] Google. Google Chrome Experiments. http://www.chromeexperiments.com/.

[30] Google. Blobstore Java API Overview, 2009. https://developers.google.com/appengine/docs/java/blobstore/.

[31] Orion Henry James Lindenbaum, Adam Wiggins. Heroku— Cloud Application Platform, 2011. https://www.heroku.com.

[32] Daniel Lee. Display a UML diagram using Draw2D. *IBM. Retrieved*, 26(01):2008, 2003.

[33] Avraham Leff and James T Rayfield. Web-application development using the model/view/controller design pattern. In *Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International*, pages 118–127. IEEE, 2001.

[34] Liebman. Slickgrid Wiki. GitHub. `https://github.com/mleibman/SlickGrid/wiki`.

[35] Brian Miller. SlickGrid - Versatile, Customizable, FAST. `http://www.endertech.com/blog/slickgrid-versatile-customizable-fast`.

[36] Mark Pilgrim. *HTML5: Up and running*. O'Reilly, 2010.

[37] Antoine Quint. Scalable vector graphics. *Multimedia, IEEE*, 10(3):99–102, 2003.

[38] Strathausen. Graph Dracula. `http://www.graphdracula.net`.